



Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Institut für Informatik

Repository-Mining von BPMN-Prozessen

BACHELORARBEIT

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
im Studiengang Informatik

eingereicht von Viktor Stefanko
geb. am 18.10.1991 in Serbytschany

Betreuer: Dr. Thomas Heinze
Themenverantwortlicher: Prof. Dr. habil. Wolfram Amme

Jena, den 31. Juli 2019

Zusammenfassung

Das Thema dieser Arbeit ist das systematische und automatisierte Durchsuchen und Analysieren von öffentlichen GitHub-Repositorys nach BPMN-Prozessen. Der zugrundeliegende allgemeine Ansatz ist in der Forschungsliteratur auch als „Mining Software Repositories (MSR)“ bekannt. Es wurde ein Programm entwickelt, mit dem durchschnittlich 2.103 GitHub-Repositorys pro Stunde durchsucht werden können. Dieses Programm kann parallel ausgeführt werden, wodurch das Durchsuchen von GitHub-Repositorys weiter beschleunigt werden kann.

Im Rahmen dieser Arbeit wurde das Mining-Programm parallel auf vier Rechnern in einem Zeitraum von 30 Tagen, 12 Stunden und 48 Minuten ausgeführt und es konnten so 10% aller öffentlichen GitHub-Repositorys analysiert werden. Es wurden dabei insgesamt 6.163.217 Repositorys durchsucht und in 1.251 Repositories (0,02%) konnten BPMN-Prozesse gefunden werden.

Die gefundenen BPMN-Prozesse wurden hinsichtlich verschiedener Merkmale, wie beispielsweise Alter, Anzahl an Änderungen und Autoren, untersucht. Daneben wurden auch die ermittelten Repositorys hinsichtlich Merkmalen, wie beispielsweise dominante Programmiersprache und Lokalisation, analysiert. Bei diesen Analysen wurde deutlich, dass das Erscheinen der Version 2.0 von BPMN im Jahr 2011 einen großen Schub für die Verbreitung von BPMN-Prozessen auf GitHub mit sich brachte. So beträgt der durchschnittliche jährliche Zuwachs an GitHub-Repositorys mit BPMN-Prozessen seit 2011 ganze 71,7%.

Für die gefundenen BPMN-Prozesse im XML-Format wurde zusätzlich auch eine Überprüfung der durch die BPMN-Spezifikation vorgegebenen Syntax- und Semantikregeln durchgeführt. Zu diesem Zweck wurde auf das Analysewerkzeug BPMNspector zurückgegriffen. Durch die Analyse stellte sich heraus, dass der überwiegende Teil der auf GitHub gefundenen BPMN-Prozesse (83,5%) Verletzungen der Syntax- und Semantikregeln von BPMN enthält.

Um die Verletzungen gegenüber der BPMN-Spezifikation zu korrigieren, wurde das Reparierwerkzeug BPMNspector-fixSeqFlow angewendet, mit dem 3.270 BPMN-Prozesse (36,7%) repariert werden konnten.

Inhaltsverzeichnis

Zusammenfassung	3
Abkürzungsverzeichnis	5
1. Einleitung	6
1.1. Einführung und Problemstellung	6
1.2. Ziele	6
1.3. Aufbau der Arbeit	7
2. Grundlagen	8
2.1. BPMN	8
2.1.1. Elemente von BPMN	9
2.1.2. Beispielprozess	13
2.2. Mining Software Repositorys	15
3. Mining von BPMN-Prozessen	17
3.1. Ansätze	17
3.1.1. GHTorrent	17
3.1.2. Klonen	18
3.1.3. GitHub API	19
3.1.3.1. Suche via GitHub API	19
3.1.3.2. Durchsuchen der Baumstruktur	20
3.2. Implementierung des Mining-Prozesses	21
3.2.1. Datenbank	22
3.2.2. Das Mining-Programm	24
4. Analyse gefundener BPMN-Dateien	31
4.1. Resultate des Mining-Programms	31
4.2. Allgemeine Statistiken	32
4.2.1. Statistiken über GitHub-Repositorys	33
4.2.1.1. Dominante Programmiersprache	33
4.2.1.2. Erstellungsjahre und Jahre der letzten Änderung	34
4.2.1.3. Dauer der Aktivität	36
4.2.1.4. Anzahl der Commits	37
4.2.1.5. Standorte der Repository-Entwickler	37
4.2.2. Statistiken über gefundene BPMN-Dateien	38
4.2.2.1. Analyse der Dateien bezüglich Duplikate	38
4.2.2.2. Grafische Repräsentation der BPMN-Prozesse im XML-Format	40

4.2.2.3.	Größe der BPMN-Prozesse im XML-Format	41
4.2.2.4.	Autoren	42
4.2.2.5.	Dateialter	42
4.2.2.6.	Dateiänderungen	43
4.3.	Statistiken über Korrektheit der BPMN-Prozesse im XML-Format . .	44
4.3.1.	Analyse mit dem Programm BPMNspector	44
4.3.2.	Reparieren mit dem Programm BPMNspector-fixSeqFlow . .	46
5.	Fazit	50
6.	Ausblick	52
	Literaturverzeichnis	56
	Abbildungsverzeichnis	58
	Tabellenverzeichnis	59
	Anhang	59
A.	Anhang	60
A.1.	Die 4 am meisten duplizierten BPMN-Dateien im XML-Format . . .	60
B.	Anhang	62
B.1.	Die 5 größten BPMN-Prozesse im XML-Format	62
C.	Anhang	68
C.1.	Typen von Verstößen gegen die Syntax- und Semantikregeln von BPMN 2.0	68

Abkürzungsverzeichnis

API	Application Programming Interface
BPE	Business Process Engine
BPM	Business Process Managemen
BPMN	Business Process Model and Notation
CSS	Cascading Style Sheets
EPK	Ereignisgesteuerten Prozesskette
EXT	Extended Constraints
GB	Gigabyte
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ISO	Internationale Organisation für Normung
JSON	JavaScript Object Notation
KB	Kilobyte
MSR	Mining Software Repositorys
OMG	Object Management Group
SSH	Secure Shell
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema Definition

1. Einleitung

1.1. Einführung und Problemstellung

Die Business Process Model and Notation (BPMN) ist der führende Standard im Bereich der Geschäftsprozesse und Workflow-Modellierungssprachen [49]. Sie wurde im Jahre 2013 als Standard für die Prozessmodellierung von der Internationalen Organisation für Normung (ISO) festgeschrieben.

Nach Meinung von vielen IT-Spezialisten spiegelt sich die Nutzung und Verbreitung von Programmiersprachen und -techniken in öffentlichen Software-Repositorys wieder, die somit als Indikator für Trends in der ganzen IT-Branche gesehen werden können. Obwohl die Bedeutung von BPMN für die Geschäftsprozessmodellierung seit Einführung der Version 2.0 deutlich gewachsen ist, gibt es bislang noch keine Arbeiten, die die Nutzung und Verbreitung von BPMN in öffentlichen Software-Repositorys, wie beispielsweise GitHub, betrachten.

1.2. Ziele

In dieser Arbeit sollen zunächst die Grundlagen von BPMN und dem Forschungsgebiet "Mining Software Repositorys (MSR)" aufgearbeitet und beschrieben werden. Es sollen dazu auch die möglichen Ansätze für die Suche nach BPMN-Prozessen in GitHub beschrieben und diskutiert werden. Anschließend muss das von uns entwickelte Mining-Programm für die Suche und Analyse von BPMN-Prozessen auf GitHub vorgestellt werden.

Es sollen weiterhin die Ergebnisse der Analyse von 10% aller öffentlichen GitHub-Repositorys nach BPMN-Prozessen präsentiert werden und anschließend nach verschiedenen Merkmalen analysiert werden. Bei der Analyse der GitHub-Repositorys mit BPMN-Prozessen soll ermittelt werden, wie groß die durchschnittliche Aktivitätsdauer der Repositorys ist und wie viele Commits im Durchschnitt gemacht werden. Es soll auch untersucht werden, welche Zusammenhänge es zwischen der Benutzung von BPMN-Prozessen und Programmiersprachen bzw. Standorten der GitHub-Benutzer gibt.

Bei der Analyse der gefundenen BPMN-Dateien soll ermittelt werden, wie groß der Anteil der Duplikate unter allen gefundenen Dateien ist. Die gefundenen BPMN-Dateien im XML-Format sollen hinsichtlich der grafischen Representation und Anzahl der XML-Elemente untersucht werden.

Die gefundenen BPMN-Prozesse im XML-Format sollen mit dem Analysewerk-

zeug BPMNspector gegenüber Verstößen der Syntax- oder Semantikregeln von BPMN 2.0 analysiert werden. Die Dateien, in denen die Verletzungen von Syntax- oder Semantikregeln der BPMN-Spezifikation gefunden wurden, sollen mit dem Reparierwerkzeug BPMNspector-fixSeqFlow korrigiert werden und mit BPMNspector wiederholt untersucht werden.

1.3. Aufbau der Arbeit

Diese Arbeit besteht aus folgenden Kapiteln:

Kapitel 2 - Grundlagen

Hier werden Grundlagen über die BPMN-Prozesse und das Forschungsgebiet Mining Software Repositorys (MSR) präsentiert.

Kapitel 3 - Mining von BPMN-Prozessen

In diesem Kapitel werden zuerst Ansätze für die Suche nach BPMN-Prozessen auf GitHub analysiert und danach Struktur und Aufbau des zu diesem Zweck implementierten Mining-Programms beschrieben.

Kapitel 4 - Analyse gefundener BPMN-Prozesse

Im vierten Kapitel werden die Resultate der Programmausführung dargestellt und die gefundenen BPMN-Prozesse und ihre Repositorys analysiert. Danach werden die Statistiken über die Korrektheit der BPMN-Prozesse im XML-Format, die mithilfe des Programms BPMNspector gebildet wurden, präsentiert. Anschließend werden die Ergebnisse der Anwendung von Reparierwerkzeug BPMNspector-fixSeqFlow und wiederholter Anwendung von BPMNspector dargestellt.

Kapitel 5 - Fazit

Hier wird die Arbeit zusammengefasst und es werden Schlüsse aus den Ergebnissen der Arbeit gezogen.

Kapitel 6 - Ausblick

Hier, im letzten Kapitel, werden die möglichen Erweiterungen von unserer Forschung beschrieben.

2. Grundlagen

Das folgende Kapitel gibt einen kurzen Überblick über die Geschäftsprozessmodellierungssprache BPMN und das systematische Auswerten und Durchsuchen von Software Repositorys, welches als Forschungsgebiet unter dem Begriff Mining Software Repositorys (MSR) zusammengefasst wird.

2.1. BPMN

Die Business Process Model and Notation (BPMN) [16] ist eine Spezifikation, die eine grafische Notation zur Modellierung von Geschäftsprozessen und ein XML-Format zur Prozessserialisierung bereitstellt. Sie ist ein öffentlicher Standard, der von der Object Management Group (OMG) [36] verwaltet wird. Das gesamte Dokument, das BPMN ausführlich beschreibt, umfasst mehr als fünf hundert Seiten und kann von der offiziellen Internetseite der OMG heruntergeladen werden (siehe [16]).

Das Hauptziel von BPMN, wie in der Spezifikation [36] benannt, ist es, eine Notation bereitzustellen, die für alle Anwender leicht verständlich und zu verwenden ist. Dies beinhaltet sowohl die für die Prozessmodellierung verantwortlichen Domänenexperten, wie auch die IT-Experten, die die Prozesse im Anschluss implementieren. Somit schafft BPMN eine standardisierte Brücke für die Lücke zwischen Geschäftsprozessdesign und Prozessimplementierung [37].

Grundsätzlich basiert BPMN auf der Überarbeitung und Anpassung anderer Notationen und Methoden für die Geschäftsprozessmodellierung, insbesondere des Aktivitätsdiagramms der Unified Modeling Language (vereinheitlichte Modellierungssprache), kurz UML [50], des Aktivitäts-Entscheidungsflussdiagramms und der Ereignisgesteuerten Prozesskette (EPK) [39], [34]. Die aktuelle Version von BPMN (2.0.2) definiert unter anderem das XML-basierte Format für das Speichern und die Übertragung von BPMN-Prozessen.

BPMN-Prozesse können mithilfe einer Business Process Engine (BPE) direkt ausgeführt werden [16]. Eine weit verbreitete Business Process Engine ist beispielsweise Camunda BPM [12], ein Java-basiertes System zur Workflow- und Prozessautomatisierung, das auch BPMN unterstützt. Dieses System wurde von der in Berlin ansässigen Firma Camunda entwickelt und besteht aus einer Reihe von Komponenten. Eine von diesen Komponenten, Camunda Modeler [17], wird für die Erstellung und das Editieren von BPMN-Prozessen verwendet. Dieses Programm kann als Desktop-Anwendung oder als Eclipse-Plugin benutzt werden. Camunda Modeler hat zwei Editoren: einen grafischen Editor und einen XML-basierten Texteditor.

Die Abbildungen 2.1 und 2.2 zeigen die beiden Editoren bei der Erstellung eines BPMN-Prozessdiagramms, das am Ende dieses Abschnitts in der Abbildung 2.4 präziser dargestellt und beschrieben ist.

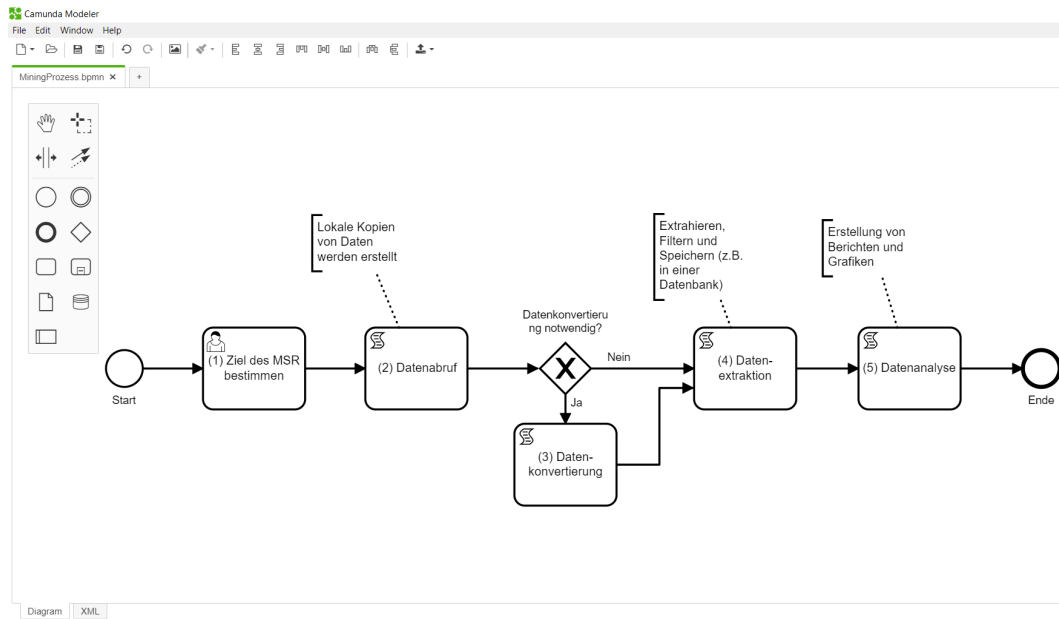


Abbildung 2.1.: Camunda Modeler (grafischer Editor)

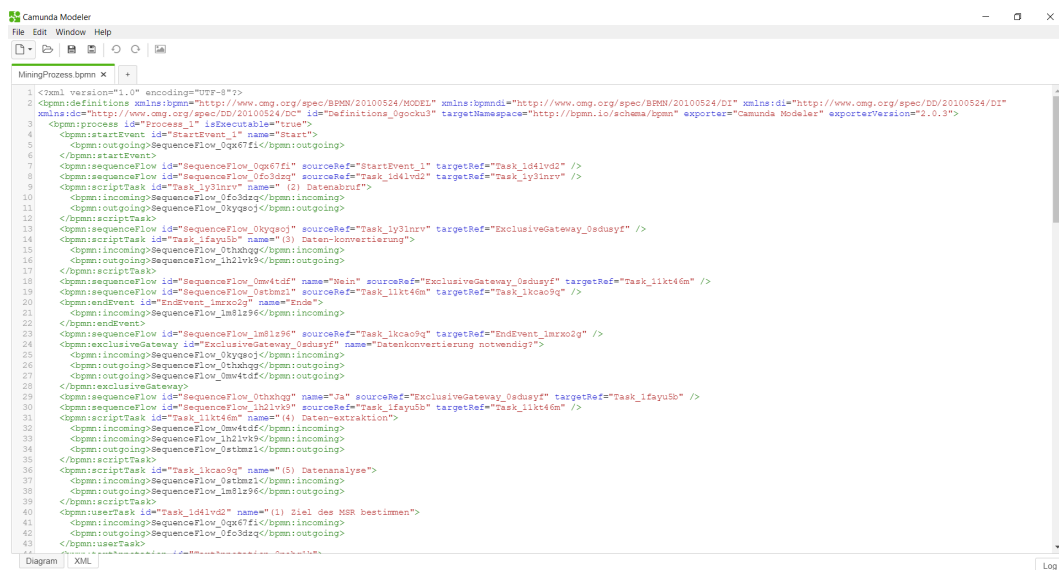


Abbildung 2.2.: Camunda Modeler (Texteditor)

2.1.1. Elemente von BPMN

Insgesamt werden in der Spezifikation von BPMN vier grundlegende Arten von Prozesstypen definiert:

- Prozessdiagramm (Process Diagram)
- Kollaborationsdiagramm (Collaboration Diagram)
- Choreographie-Diagramm (Choreography Diagram)

- Konversationsdiagramm (Conversation Diagram)[37].

Für die Definition eines Prozesses, stellt die Spezifikation von BPMN eine Menge von Elementen sowie Regeln für deren Verwendung bereit. Derart werden die Syntax und Semantik von BPMN natürlichsprachlich definiert [35]. Jedes BPMN-Element kann dabei einer von fünf grundsätzlichen Elementkategorien zugeordnet werden [16]. Diese sind:

1. **Flow Objects** (Knoten)
2. **Data** (Daten)
3. **Connecting Objects** (Verbindende Objekte)
4. **Swimlanes** (Schwimmbahnen, die Teilnehmer darstellen)
5. **Artifacts** (Artefakte)

In der Tabelle 2.1 werden die wichtigsten Elemente von BPMN-Prozessen kurz erläutert sowie ihre visuelle Notation angegeben.

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen [16], [35], [1]

Element	Beschreibung	Grafische Notation
Event (Ereignis)	<p>Ereignisse geben an, wann- und was für ein Ereignis auftritt. Die Ereignisse können in 4 Kategorien eingruppiert werden, je nachdem, wann sie den Prozessablauf beeinflussen:</p> <ul style="list-style-type: none"> • Startereignis (Start Event) bezeichnet den Auslöser eines Prozesses. • Zwischenereignisse (Intermediate Events) bezeichnen Ereignisse, die während der Prozessabarbeitung eintreten können. • Grenzereignisse (Border Events) treten während der Abarbeitung einer Aktivität auf und ändern den Prozessfluss. • Endereignis (End Event) bezeichnet das Ende eines Prozesses. <p>Der Typ des Ereignisses wird durch ein Symbol in der Mitte des Kreises bestimmt, siehe Abbildung 2.3.</p>	<p>The diagram illustrates four types of BPMN events: 1. Start Event: A thick circle. 2. Intermediate Event: A circle with a double circle in the middle. An example shows 'Task A' followed by this event, which then leads to 'Task B'. 3. Border Event: A circle with a rectangle on one side. An example shows a rectangle with this event on its bottom side, with an arrow pointing down from the event. 4. End Event: A thick circle with a thick border.</p>

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen





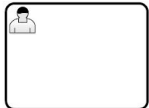

Element	Beschreibung	Grafische Notation
Activity (Aktivität)	<p>Eine Aktivität ist ein Oberbegriff für die Arbeitsschritte, die in einem Prozess ausgeführt werden. Eine Aktivität wird als Rechteck mit abgerundeten Ecken gezeichnet und kann atomar oder zusammengesetzt sein. Eine atomare Aktivität ist eine Aufgabe (Task), die beschreibt, was zu erledigen ist. Eine zusammengesetzte Aktivität ist ein Unterprozess (Subprocess), der aus anderen BPMN-Elementen bestehen kann.</p> <p>Zu den meistverbreiteten Aufgabentypen (Tasks) gehören folgende:</p> <ul style="list-style-type: none"> • Empfang einer Nachricht (Receive Task) ist eine Alternative zu einem Ereignis, welches das Eintreffen einer Nachricht modelliert. • Skript-Aktivität (Script Task) bezeichnet die Ausführung eines Skripts. Ein Skript ist meist ein kleines Programm in einer Programmiersprache, die durch die Business Process Engine unterstützt wird (beispielsweise ein Groovy-Skript in Camunda). • Eine Service-Aktivität (Service Task) modelliert einen Arbeitsschritt, der durch einen Dienst oder eine Anwendung, beispielsweise einen Web-Service oder eine Java-Klasse, automatisch ausgeführt wird. • Benutzeraktivitäten (User Task) modellieren manuelle Arbeitsschritte und gestatten die Einbindung von Personen, beispielsweise Sachbearbeitern, in einen Geschäftsprozess. • Senden einer Nachricht (Send Task), ist eine Alternative zu einem Ereignis, welches das Senden einer Nachricht modelliert. 	     

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen






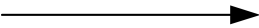
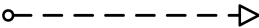
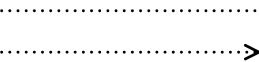

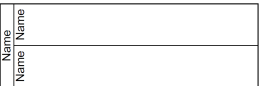
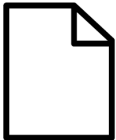


Element	Beschreibung	Grafische Notation
Gateway	<p>Ein Gateway wird verwendet, um den Kontrollfluss in einem Geschäftsprozess zu modellieren. Dabei können fünf Arten von Gateways grundsätzlich unterschieden werden:</p> <ul style="list-style-type: none"> • Datenbasiertes exklusives Gateway (XOR-Gateway) modelliert das Auseinander- oder Zusammenlaufen von alternativen Kontrollflüssen, wodurch sich Verzweigungen oder Schleifen realisieren lassen. Die Entscheidung welcher Kontrollfluss zur Ausführung kommt, wird dabei anhand von Prozessdaten getroffen. • Paralleles Gateway (AND-Gateway) modelliert das Auseinander- oder Zusammenlaufen von parallelen Kontrollflüssen. • Datenbasiertes inklusives Gateway (OR-Gateway) modelliert das Auseinander- oder Zusammenlaufen von mehreren Kontrollflüssen. Die Entscheidung welche Kontrollflüsse zur Ausführung kommen, wird dabei anhand von Prozessdaten getroffen. • Ereignisgesteuertes Gateway (Event-Gateway). Es wird wie beim XOR-Gateway nur ein Pfad gewählt. Dieser Pfad wird aber nicht anhand von Daten, sondern anhand des nächsten Ereignisses bestimmt. • Komplexes Gateway (Complex-Gateway) wird zum Modellieren eines komplexen Synchronisationsverhaltens verwendet. 	    

Tabelle 2.1.: Grundlegende Modellierungselemente von BPMN-Prozessen

Element	Beschreibung	Grafische Notation
Sequence Flow (Sequenzfluss)	Ein Sequenzfluss wird verwendet, um die Reihenfolge zu definieren, in der Aktivitäten ausgeführt werden.	
Message Flow (Nachrichtenfluss)	Ein Nachrichtenfluss wird verwendet, um den Nachrichtenfluss zwischen Prozessteilnehmern zu modellieren.	
Association (Assoziation)	Eine Assoziation wird verwendet, um Informationen und Artefakte, beispielsweise Datenobjekte, mit anderen BPMN-Elementen, beispielsweise Aktivitäten, zu verknüpfen.	
Pool (Schwimmbecken)	Ein Pool ist die grafische Darstellung eines Teilnehmers an einer Kollaboration. Er wird auch als Swimlane bezeichnet und fungiert als grafischer Container, um eine Gruppe von Aktivitäten von anderen Pools zu trennen.	
Lane (Spur)	Eine Spur ist eine Unterteilung innerhalb eines Prozesses, manchmal innerhalb eines Pools, und erstreckt sich über die gesamte Länge des Prozessdiagramms, entweder vertikal oder horizontal.	
Data Object (Datenobjekt)	Datenobjekte geben an, welche Informationen von Prozessaktivitäten produziert und/oder konsumiert werden. Datenobjekte können zwischen Aktivitäten ausgetauscht werden und definieren dadurch Datenflüsse im Prozess.	
Message (Nachricht)	Eine Nachricht wird verwendet, um den Austausch von Informationen zwischen zwei Teilnehmern, typischerweise über Prozessgrenzen hinweg, darzustellen. Der Nachrichtenaustausch kann dabei sowohl synchron als auch asynchron erfolgen.	
Text Annotation (Textanmerkung)	Textanmerkungen sind ein Mechanismus für einen Modellierer, um dem Leser eines BPMN-Diagramms zusätzliche Textinformationen, beispielsweise Kommentare, bereitzustellen.	

2.1.2. Beispielprozess

Wir möchten uns in der Abbildung 2.3 ein einfaches BPMN-Prozess anschauen, das den Prozess der Buchausleihe aus Bibliothek modelliert.

An dem Prozess „Buchausleihe“ sind eine Bibliothek als Institution und ein Bibliotheksbenutzer beteiligt. Der Prozess wird vom Benutzer gestartet, der zuerst ein Buch für die Ausleihe auswählt und die Buchausleihe beantragt. Die Bibliothek kann die Ausleihe bestätigen oder absagen. Im Falle der Bestätigung wird das Buch ausgeliehen und im Falle der Absage wird dem Antragsteller der Grund der Absage mitgeteilt. Der Benutzer kann das Buch vor Ende oder am Ende der Leihfrist der Bibliothek zurückgeben. Der Prozess ist zu Ende, wenn die Bibliothek das Buch zurückbekommt.

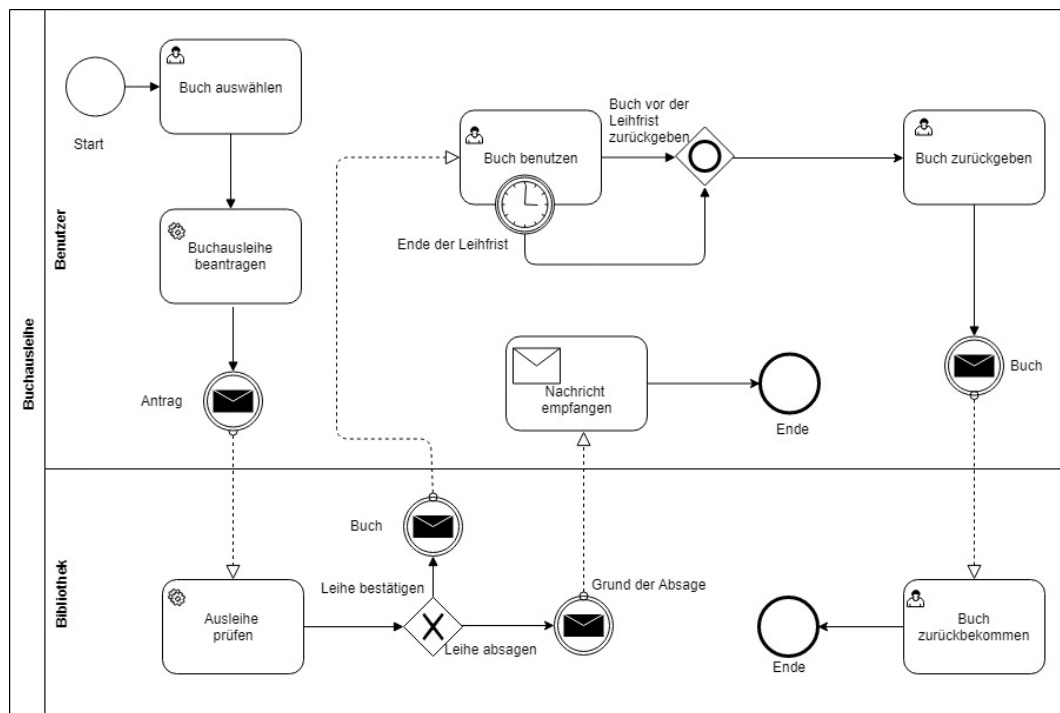


Abbildung 2.3.: Buchausleihe

Jetzt versuchen wir die Abbildung 2.3 zu beschreiben und dabei die dargestellten Elemente zu benennen. Insgesamt besteht dieses Diagramm aus folgenden Elementen:

- Schwimmbcken.
- Spur.
- Startereignis.
- Zwischenereignis: Message Throw Event.
- Grenzerereignis: Timer Catch Event.
- Aktivitäten: Benutzeraktivität und Service-Aktivität.
- Gateways: XOR-Gateway und OR-Gateway.
- Sequenzfluss.
- Nachrichtenfluss
- Endereignis.

Der gesamte Prozess befindet sich innerhalb des Pools und hat den Namen

„Buchausleihe“. Der Pool ist in zwei Spuren geteilt. Die Teilung in Spuren macht deutlich, welche Aktivitäten von der Bibliothek als Institution und welche von dem Bibliotheksbenutzer ausgeführt werden. Der Prozess startet mit dem Startereignis. Die vom Startereignis ausgehende Kante (Sequenzfluss) zeigt, welches Element als Nächstes durchlaufen wird. So werden zuerst die Aktivitäten „Buch auswählen“, „Buchausleihe beantragen“ vom Bibliotheksbenutzer und „Ausleihe prüfen“ von der Bibliothek erledigt. Danach wird ein XOR-Gateway passiert, das prüft, ob die Buchausleihe möglich ist. Im Falle der Absage wird eine Nachricht gesendet und so dem Bibliotheksbenutzer der Grund der Absage mitgeteilt. Im Falle der Zusage bekommt der Antragsteller das von ihm ausgeliehene Buch.

Die Nutzung des Buches durch den Bibliotheksbenutzer beginnt sobald das Buch empfangen wird. So kommen wir zur Aktivität „Buch benutzen“. Diese Aktivität hat ein Grenzereignis (Timer Catch Event), das eintrifft, wenn die Leihfrist endet. In diesem Fall wird ein OR-Gateway erreicht. Dieses Gateway kann auch über eine direkte Kante von der Aktivität „Buch benutzen“ erreicht werden. In diesem Fall entscheidet der Benutzer über die Zurückgabe des Buches vor der geltenden Leihfrist. Das OR-Gateway hat nur eine auslaufende Kante, die zur Aktivität „Buch zurückgeben“ führt, wo das Buch der Bibliothek zurückgegeben wird. Der gesamte Prozess endet, sobald die Bibliothek das ausgeliehene Buch zurückbekommt.

In der Abbildung fällt auf, dass es zwei Typen von Aktivitäten benutzt wurden: Service- und Benutzeraktivität. Beim Bezeichnen einer Aktivität als Benutzeraktivität wurde angenommen, dass diese „manuell“ von einer Person erledigt wird, z.B. die Aktivität "Buch auswählen". Bei Service-Aktivität wurde angenommen, dass die Bibliothek über ein Webservice verfügt, über das die Buchausleihe vom Benutzer beantragt und von der Bibliothek bestätigt werden kann.

2.2. Mining Software Repositorys

Das Forschungsgebiet Mining Software Repositorys (MSR) analysiert die heute in Software-Repositorys vielfältig vorliegenden Daten. Auf diese Weise lassen sich interessante Rückschlüsse auf Softwareentwicklungsprozesse ziehen. Zudem können Aussagen zu Prozessen und der aus diesen resultierenden Software empirisch untersucht und belegt werden.

Als wichtige Quellen für das MSR sind heutzutage Git-Versionsverwaltungssysteme GitHub [26] und GitLab [29]. Diese Webdienste zur Arbeit mit Git enthalten Millionen von Softwarerepositorys, in denen Änderungen an Dateien über die Zeit hinweg protokolliert sind. Man kann nicht nur auf eine bestimmte Version der Dateien zurückgreifen, sondern es ist möglich, die im Versionsverwaltungssystem vorliegende Daten automatisiert abzurufen und zu analysieren. Wie zum Beispiel:

- Der Quellcode von Programmen. Dies ist die wichtigste Eingabe für die mögliche Analyse.

- Durch die während der Ausführung der Software gesammelten Daten können Profile erhalten werden, die mitteilen, welche Teile der Software häufig verwendet werden und welche nicht.
- Den Produkten liegen möglicherweise zusätzliche Dokumentationen bei, beispielsweise: Wikis oder Issue Tracker (siehe GitHub/GitLab). Diese können auch wichtige Funktionen enthalten, die erklären, warum Code so aussieht, wie er aussieht.
- Die resultierende Software kann statisch analysiert werden und bietet Funktionen wie Komplexitätsmetriken oder Abhängigkeiten. Die Ergebnisse von solchen statischen Analysen, aber auch anderen Analysen und Tests, finden sich oft in Continuous Integration Systemen (CI-Systemen) wieder.
- Versionsarchive zeichnen die am Produkt vorgenommenen Änderungen auf, einschließlich wer, wann, wo und warum die Änderungen durchgeführt hat [33].

Die Abbildung 2.4 zeigt den allgemeinen Algorithmus für das MSR in Form eines BPMN-Prozesses.

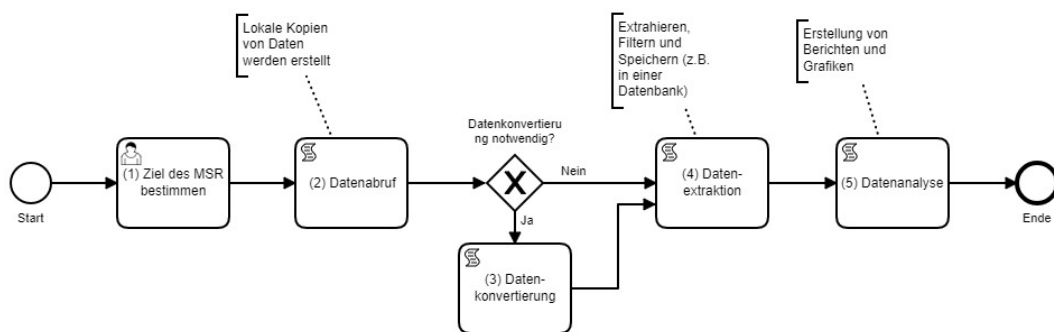


Abbildung 2.4.: Algorithmus für das MSR

1. **Ziel des MSR bestimmen**, d.h. welche Daten sind zur Beantwortung der Fragestellungen oder zur Validierung der Hypothesen notwendig.
2. **Datenabruf**. Um die ständige Verfügbarkeit der gezielten Daten sicherzustellen und einen schnellen Zugriff auf sie zu ermöglichen, sollten lokale Kopien erstellt werden.
3. **Datenkonvertierung** (optional). Data Mining erfordert, dass die Daten nicht nur heruntergeladen und verarbeitet werden, sondern auch, dass man viele ihrer Merkmale im Voraus versteht. Daher sollten die Daten den Voraussetzungen und Annahmen entsprechen, die vorher getroffen wurden.
4. **Datenextraktion**. Nachdem die Daten auf lokaler Festplatte in einer geeigneten Form gesichert wurden, können Sie verarbeitet werden. Die Verarbeitung umfasst das Extrahieren, Filtern und Speichern. Zum Beispiel können die gefilterten Informationen in einem relationalen Datenbankmanagementsystem gespeichert werden.
5. **Datenanalyse**. In diesem Schritt wird die gezielte Analyse der gespeicherten Daten durchgeführt [33].

3. Mining von BPMN-Prozessen

Das folgende Kapitel erläutert Ansätze für die Suche nach BPMN-Prozessen auf GitHub und die Struktur des zu diesem Zweck implementierten Mining-Programms.

3.1. Ansätze

In der ersten Phase der Bachelorarbeit wurden verschiedene Ansätze für die systematische Suche nach GitHub-Repositorys, die BPMN-Prozesse enthalten, untersucht und getestet. Grundlage und Inspiration dafür bildeten Publikationen zum Repository-Mining von UML-Diagrammen [22], [32], [44], [43]. Die Ansätze sollen im Folgenden kurz vorgestellt werden.

3.1.1. GHTorrent

Entsprechend der offiziellen Quellen von GitHub verfügt diese Plattform weltweit über mehr als 31 Millionen registrierte Benutzer und mehr als 100 Millionen Repositorys [27], wobei fast ein Drittel von ihnen innerhalb des letzten Jahres erzeugt wurde [47]. Diese Zahlen zeigen, dass GitHub eine sehr dynamische Hosting-Plattform ist und einen sehr großen Bestand von Daten enthält. Um diese Daten effektiv analysieren zu können, wird eine statische, zuverlässige und fortlaufend aktualisierte Quelle zu dieser Plattform benötigt [22]. Als solche Quelle wurde von uns das GHTorrent-Projekt ausgewählt [46].

Das GHTorrent-Projekt wurde entwickelt, wie seine Gründer angeben, um das Mining von GitHub-Repositorys zu erleichtern. Dieses Projekt bietet einen skalierbaren, abfragbaren Offline-Spiegel der GitHub-Daten [31]. GHTorrent aktualisiert seine Daten jeden Monat. Die Daten können sowohl heruntergeladen als auch ohne herunterzuladen online abgefragt werden. Herunterladen kann man sie als MySQL-Dump (Menge von csv-Dateien, wobei jede von denen einer MySQL-Datenbanktabelle entspricht) oder als MongoDB-Dump. Das Abfragen ist möglich sowohl durch eine GHTorrent-Online-Abfrageoberfläche [31] oder mithilfe von Google BigQuery [30], das einen aktuellen Import des neuesten MySQL-Dumps von GHTorrent enthält [25].

Nachdem der MySQL-Dump als tar-Archiv (85,5 GByte) mit 23 unterschiedlichen csv-Dateien und MySQL-Datenbankschema heruntergeladen wurde, haben wir die Dateien „users.csv“, die Informationen über GitHub-Benutzer enthält, und „projects.csv“ mit Informationen über GitHub-Repositorys lokal in gleichnamige Da-

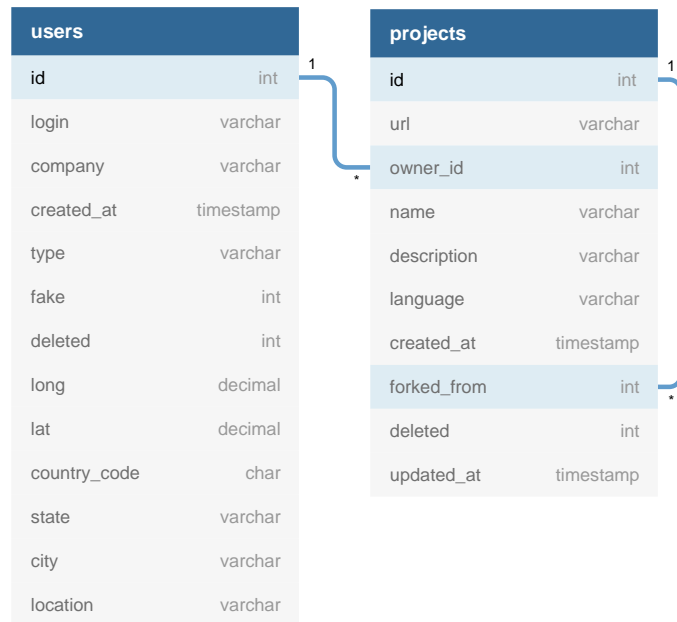


Abbildung 3.1.: MySQL-Datenbankschema der Tabellen „users“ und „projects“

tenbanktabellen importiert (siehe auch Abbildung 3.1). Auf Grundlage dieser Daten konnten wir eine Liste von nicht-gelöschten und nicht-geforkten (d.h. nicht kopierten) Repositories bilden.

GHTorrent stellt allerdings keine Informationen zu Dateien aus einem bestimmten GitHub-Repository zur Verfügung [22]. Es ist daher erforderlich, diese Informationen entweder über das Klonen und anschließende lokale Durchsuchen der Repositories oder online über Abfragen der GitHub API abzurufen.

3.1.2. Klonen

Da GitHub die verteilte Versionskontrolle `git` als Standardmechanismus zur Versionverwaltung verwendet, wäre es grundsätzlich möglich, mithilfe des `git clone` Befehls, die GitHub-Repositories herunterzuladen und dann nach BPMN-Prozessen zu durchsuchen.

Es wurde ein Python-Programm für die Suche nach Repositories mit BPMN-Prozessen geschrieben. In diesem Programm wurde in einer Schleife über die Liste mit Namen der GitHub-Repositories iteriert und bei jeder Iteration mit `git clone` ein Repository heruntergeladen und durchsucht.

Trotz der richtigen Ergebnissen, die dieses Verfahren liefert, ist es zu langsam, wenn es sich um große Repositories handelt. Dieses Problem wird auch in [45] beschrieben. Das Problem besteht darin, dass beim Klonen eines Repository nicht nur Dateien, sondern auch die ganze Geschichte des Repository heruntergeladen wird. Bei großen Repositories werden bis zu mehreren GB Daten heruntergeladen, was entsprechende Ressourcen benötigt. Deshalb stellte sich dieses Verfahren für unsere Forschung als ineffektiv heraus.

3.1.3. GitHub API

GitHub bietet eine Anwendungsprogrammierschnittstelle (englisch: Application Programming Interface, kurz API) für den Zugriff auf Daten in öffentlichen Repositories. Der gesamte API-Zugriff erfolgt über HTTPS und kann über `https://api.github.com` aufgerufen werden. Es handelt sich dabei um eine REST-Schnittstelle und alle Daten werden als JSON-Objekte gesendet und empfangen [41].

Die GitHub API definiert eine Beschränkung von 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer und 60 Anfragen pro Stunde für nicht authentifizierte. Nicht authentifizierte Anfragen werden der IP-Adresse zugeordnet und nicht dem Benutzer, der Anfragen stellt [41].

3.1.3.1. Suche via GitHub API

Die Suchschnittstelle der GitHub API ermöglicht die Suche nach Objekten in Repositories. Es können, beispielsweise, spezifische Daten (z. B. Code, Sterne, Commits oder Mitwirkende) in einem oder in mehreren Repositories gesucht werden. Allerdings gibt es folgende Beschränkungen bei der Such-API:

- Es können höchstens 30 authentifizierte oder 10 nicht authentifizierte Anfragen pro Minute gestellt werden.
- Die Antwort für eine Anfrage kann höchstens 1000 Resultate haben.
- Es wird nur der Standardzweig eines Repository (default branch) bei der Suche berücksichtigt. In den meisten Fällen ist dies der Hauptzweig (master branch).
- Nur Dateien, die kleiner als 384 KB sind, können durchsucht werden [41].

Es wurde wiederum ein Python-Programm geschrieben, in dem in einer Schleife über die Liste mit Namen der GitHub-Repositories iteriert wurde und bei jeder Iteration eine Anfrage über die GitHub API gesendet wurde.

Zum Beispiel für die Suche nach dem Schlüsselwort `bpmn` im Dateiinhalt wurde für GitHub-Benutzer `ViktorStefanko` und Repository `BPMN_Crawler` folgende Anfrage gesendet:

```
https://api.github.com/search/code?q=bpmn+in:file+user:ViktorStefanko+repo:BPMN_Crawler.
```

Die Abbildung 3.2 zeigt das zurückgelieferte JSON-Objekt. Man kann sehen, dass innerhalb des `BPMN_Crawler` Repository 6 Dateien gefunden wurden, die das Schlüsselwort `bpmn` im Dateiinhalt enthalten.

Die Suche via GitHub API hat einen großen Vorteil im Vergleich zum oben beschriebenen Vorgehen über das Klonen von Repositories: sie ist viel schneller. Allerdings gibt es auch Nachteile im Vergleich zum Klonen. Zum einen können Dateien größer als 384 KB nicht durchsucht werden. Und zum anderen können nur 1000 Resultate für ein Repository ermittelt werden. Folglich können bei sehr großen Repositories

```

1  {
2      "total_count": 6,
3      "incomplete_results": false,
4      "items": [
5          {"name": "run_bpmn_spectator.py"...},
61          {"name": "file_statistics.py"...},
157         {"name": "tree_crawler.py"...},
233         {"name": "README.md"...},
309         {"name": "schema.sql"...},
385         {"name": "main_script.py"...}
461     ]
462 }

```

Abbildung 3.2.: Beispiel von einem zurückgelieferten JSON-Objekt der Suchschnittstelle der GitHub API

nicht alle Ergebnisse ermittelt werden.

Insgesamt lassen sich mit Hilfe der Suchschnittstelle der GitHub API 1800 Repositories pro Stunden durchsuchen (30 authentifizierte HTTP-Anfragen pro 1 Minute * 60 Minuten = 1800).

3.1.3.2. Durchsuchen der Baumstruktur

Es gab noch ein Verfahren, dass das Benutzen der GitHub API möglich machte. Dessen Besonderheit ist es, dass nicht der Inhalt einer Datei in dem entsprechenden Projekt nach Schlüsselwörtern durchsucht wird, sondern es wird zuerst die Repositorystruktur als Baum mit Dateinamen und Ordnern bestimmt und nur in Dateinamen einschließlich der Dateiendung nach Schlüsselwörtern gesucht [22].

Dieses Verfahren besteht aus drei Schritten:

- Ermitteln des letzten `commit`, das dem Standardzweig eines Repository gehört. Dafür wird eine authentifizierte HTTP-Anfrage an die GitHub API mit (`branches/master`) als Parameter gesendet. Falls `master` branch der Standardzweig, also `default branch` ist, enthält das Ergebnis der Anfrage das letzte `commit`, was das Ziel war.

Es kann allerdings vorkommen, dass `master` branch nicht der Standardzweig (`default branch`) ist. In diesem Fall wird die zweite Anfrage gesendet, um den Standardzweig zu ermitteln. Es sei dies der Standardzweig `defBr`, dann enthält die dritte Anfrage die gesendet wird (`branches/defBr`) als Parameter. So wird auch hier das letzte `commit` ermittelt.

- Senden einer authentifizierten HTTP-Anfrage, um die Baumstruktur des jeweiligen Repository anhand des letzten `commit` zu erhalten.
- Suchen in Blättern des Baums, die Dateinamen des jeweiligen Repository sind, nach Schlüsselwörtern. Falls gefunden, Speichern des Dateipaths in ei-

ner Datenbank.

Dieses Verfahren verbindet die Vorteile des Klonens von Repositories und der Suche via GitHub API. Somit es ist schnell und es werden alle Repositories und ihre Dateinamen unabhängig von ihren Größen durchsucht.

Wegen der Beschränkung der GitHub API, können bei diesem Ansatz höchstens 5000 authentifizierte HTTP-Anfragen gestellt werden. Für jedes Repository werden mindestens zwei Anfragen gestellt: erste fürs Ermitteln des letzten `commit` und zweite fürs Ermitteln der Baumstruktur. Es können aber zusätzlich noch zwei Anfragen gebraucht werden. Das ist dann der Fall, wenn der Hauptzweig nicht `master branch` ist. Aus diesen Gründen können bei diesem Verfahren theoretisch mindestens 1250 und höchstens 2500 GitHub-Repositories pro Stunde durchsucht werden.

Bei der Umsetzung des Ansatzes stellte sich heraus, dass die Mehrheit von GitHub-Repositories `master branch` als Standardzweig hat. Daher war die Anzahl der GitHub-Repositories, die pro Stunde durchsucht werden konnten, nah zur oberen Schranke mit 2500 GitHub-Repositories. Somit war dieser Ansatz auch in diesem Sinne effektiver als das Klonen und die Suche via GitHub API.

3.2. Implementierung des Mining-Prozesses

Nach der Analyse verschiedener Ansätze haben wir GHTorrent, als Quelle für grundlegende Informationen zu GitHub-Repositories wie Name und Status ausgewählt. Zusätzlich konnte über diese Ressource auch der GitHub-Nutzer, der das Repository angelegt hat, bestimmt werden. Weil GHTorrent aber keine Informationen zu den in einem Repository enthaltenen Dateien liefert, wurde zusätzlich der in Abschnitt 3.1.3.2 beschriebene Ansatz zum Durchsuchen der Repositorybaumstruktur mit Hilfe der GitHub API verwendet.

Die Arbeit am Programm startete am Anfang des Novembers 2018. Es wurde der GHTorrent MySQL-Dump von 01.11.2018 heruntergeladen und daraus die MySQL-Datenbanktabellen „users“ und „projects“ (siehe Abbildung 3.1) extrahiert. Diese zwei Tabellen enthalten Daten, die es möglich machen, eine Liste von nicht-gelöschten und nicht-geforkten GitHub-Repositories zu bilden. Die SQL-Anfrage dafür sieht wie folgt aus:

```
SELECT users.login, projects.name FROM users, projects
WHERE projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL.
```

Das Ergebnis dieser Anfrage liefert die Namen und Zugangsdaten zu 61.632.173 GitHub-Repositories. Wegen der Beschränkung der GitHub API, die höchstens 5000 authentifizierte HTTP-Anfragen pro Stunde erlaubt, würde das Durchsuchen dieser GitHub-Repositories im bestem Fall (siehe obere Schranke in 3.1.3.2) 1.027 Tage dauern. Da dies nicht umsetzbar war, wurde die Entscheidung getroffen, nur 10 Prozent von der gesamten Anzahl, d.h. 6.163.217 GitHub-Repositories, zu durchsu-

chen. Diese Anzahl wiederum konnte im besten Fall in 103 Tage durchsucht werden. Dies erfolgte analog der Arbeit [32], da dort auch (zunächst) eine Beschränkung auf 10% von GitHub erfolgte.

Um das Durchsuchen der GitHub-Repositorys zu beschleunigen, sollte das Programm so entwickelt werden, dass es auf einem Rechner mehrfach gestartet werden kann. Es sollte also so programmiert werden, dass mehrere Instanzen des Programms mit unterschiedlichen Authentifizierungsdaten für GitHub initialisiert werden und gleichzeitig unterschiedliche Repositorys durchsucht werden.

3.2.1. Datenbank

Die Abbildung 3.3 zeigt uns die Struktur der verwendeten Datenbank, die als SQLite-Datenbank implementiert wurde.

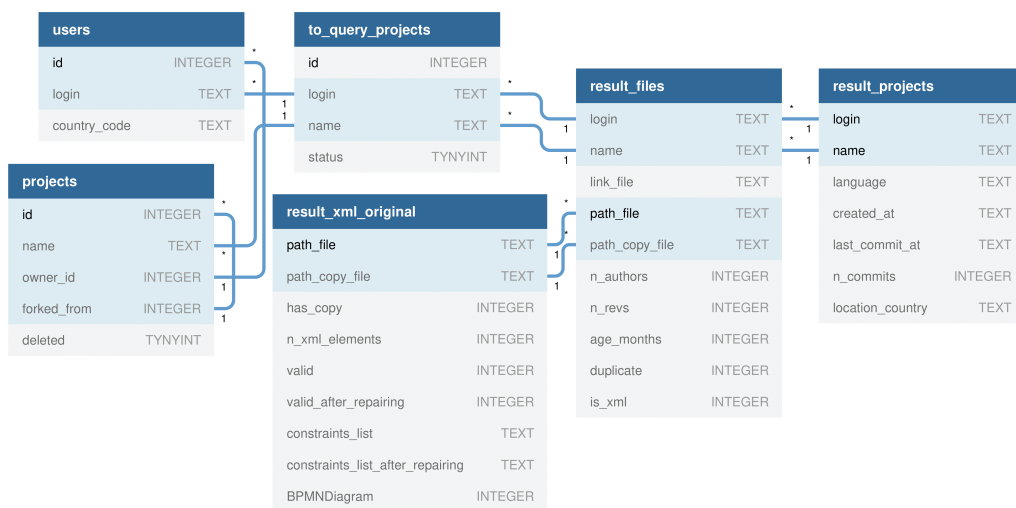


Abbildung 3.3.: Datenbankschema

Die **users** Tabelle enthält Informationen über GitHub-Nutzer und zwar Zugangsdaten (*login*) und Ländercode (*country_code*).

Die Tabelle **projects** beschreibt ein GitHub-Repository. Sie enthält Name des Repository (*name*), Verweis auf Inhaber (*owner_id*) in Tabelle **users**, Attribut *forked_from*, in dem gespeichert wird, von welchem Repository es abgeleitet wurde, wenn das überhaupt der Fall ist, und Attribut *deleted*, das angibt, ob das Repository gelöscht wurde.

Die Inhalte der **users** und **projects** Tabellen wurden aus den gleichnamigen Tabellen der MySQL-Datenbank, die aus dem GHTorrent-Archiv stammt, entnommen, siehe auch Abbildung 3.1.

Die Tabelle **to_query_projects** dient als Eingabequelle für das Mining-Programm. Sie enthält Zugangsdaten (*login*) und Name (*name*) von 10 Prozent der nicht-gelöschten und nicht-geforkten GitHub-Repositorys. Im Attribut *status* wird ge-

speichert, ob das Repository durchsucht wurde (Wert 1) oder noch zu durchsuchen ist (Wert 0). Die Daten wurden in diese Tabelle randomisiert aus **users** und **projects** mit folgender Anfrage eingefügt:

```
INSERT INTO to_query_projects (login, name) SELECT
users.login, projects.name FROM users, projects WHERE
projects.owner_id=users.id AND projects.deleted=0 AND
projects.forked_from is NULL ORDER BY RANDOM() LIMIT 6163217.
```

Die Tabelle **result_files** enthält Informationen über Dateien, die das Mining-Programm als potentielle BPMN-Prozesse eingestuft hat. Sie enthält folgende Attribute: Zugangsdaten des Inhabers (*login*), Name vom GitHub Repository (*name*), Weblink (*link_file*), Pfad innerhalb des Repository (*path_file*), Name der lokalen Kopie der Datei *copy_path_file*, Anzahl der Autoren (*n_authors*) und Änderungen (*n_revs*), Alter der Datei in Monaten (*age_months*) und Attribut *duplicate*, das angibt, ob Duplikate der entsprechenden Datei gefunden wurden. In dem Attribut *is_xml* wird gespeichert, ob die Datei ein BPMN-Prozess im XML-Format ist.

Die Tabelle **result_projects** enthält Informationen über Repositories, in denen mindestens ein potentieller BPMN-Prozess gefunden wurde. Zu diesen Informationen zählen: Zugangsdaten des Inhabers (*login*), Name des GitHub-Repository (*name*), die dominante Programmiersprache des Repository (*language*), Zeiten, zu denen das Repository erstellt wurde (*created_at*) und das letzte Mal geändert wurde (*last_commit_at*) und Anzahl der Commits (*n_commits*). Im Attribut (*location_country*) werden Namen der Länder gespeichert, aus denen die Beitragenden des Repository stammen.

Die Tabelle **result_xml_original** enthält Informationen über die BPMN-Prozesse im XML-Format. BPMN-Prozesse, die Kopien (Duplikate) haben, kommen in dieser Tabelle nur einmal vor. Die Attribute *path_file* und *path_copy_file* referenzieren die gleichnamigen Attribute aus der Tabelle **result_files** und haben die gleiche Bedeutung. Im Attribut **has_copy** wird gespeichert, ob der BPMN-Prozess Kopien hat und im Attribut **n_xml_elements** wird die Größe des BPMN-Prozesses gespeichert. Die Attribute *valid* und *valid_after_repairing* enthalten Informationen über die Validierung vom BPMN-Prozess gegenüber der Sprachspezifikation (vergleiche folgenden Abschnitt 3.2.2). Das erste von diesen Attributen enthält Informationen, die vor der Anwendung des Programms BPMNspector-fixSeqFlow gesammelt wurden, und das zweite Attribut enthält Informationen, die nach der Anwendung dieses Programms gesammelt wurden. In Attributen *constraints_list* und *constraints_list_after_repairing* werden Listen von Regelverletzungen für einen Prozess gespeichert, die bei der Validierung eventuell gefunden wurden. Analog zu den Attributen *valid* und *valid_after_repairing* enthalten *constraints_list* und *constraints_list_after_repairing* Informationen, die vor und nach der Anwendung des Programms BPMNspector-fixSeqFlow gesammelt wurden. Das Attribut *BPMNDiagram* enthält Information, ob der BPMN-Prozess grafisch darstellbar ist.

3.2.2. Das Mining-Programm

Das Mining-Programm, das mit der Programmiersprache Python programmiert wurde, kann auf GitHub unter folgendem Link: https://github.com/ViktorStefanko/BPMN_Crawler/ eingesehen werden [13].

Die Abbildung 3.4 zeigt uns den gesamten Prozess, der durch das Mining-Programm realisiert wird. Er besteht aus drei Schritten:

- (1) Im ersten Schritt werden potentielle BPMN-Prozesse aus öffentlichen GitHub-Repositorys gesammelt.
- (2) Im zweiten Schritt wird aus der Menge der potentiellen BPMN-Prozesse eine Untermenge der BPMN-Prozesse im XML-Format bestimmt.
- (3) Im letzten Schritt werden statistische Informationen über BPMN-Prozesse im XML-Format, potentielle BPMN-Prozesse und Repositorys, wo sie sich befinden, ermittelt und in der Datenbank gespeichert.

Der oben genannte erste Schritt entspricht dem Datenabruf und der zweite und der dritte Schritt implementieren die Datenextraktion des allgemeinen Algorithmus für MSR (vergleiche auch mit Abbildung 2.4).

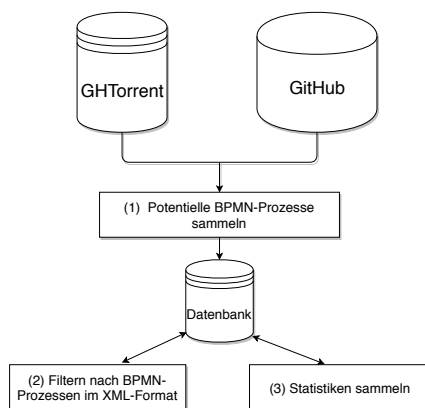


Abbildung 3.4.: Prozessdiagramm des Mining-Programms

Das Klassendiagramm in Abbildung 3.5 zeigt die Struktur des Mining-Programms. Es sind neun Klassen zu sehen. Für die jeweiligen Klassen sind die Funktionen mit ihren Rückgabetypen abgebildet. Funktionsparameter und Attribute wurden sowohl aus Platzspargründen, als auch, weil sie in diesem Kontext nicht von besonderer Bedeutung sind, weggelassen. Die Funktionen der Klassen *RepositoryCrawler* und *TreeCrawler* haben wir von Gregorio Robles entwickeltem „2016-uml-miner“-Programm [42], [22], [32], [44], [43] übernommen und für unsere Zwecke angepasst.

Die **GitHubApiCrawler** Klasse enthält die Funktion `run_api_crawler()`, die vollautomatisiert den ersten Schritt aus Abbildung 3.4 implementiert. Sie bekommt als Parameter zwei Integer Variablen, die eine untere und eine obere Schranke für das Attribut `id` von Repositorys in der `to_query_projects` Datenbanktabelle definiert, zwischen denen alle Repositorys nach BPMN-Prozessen zu durchsuchen sind. Die Schranken dienen dazu, das Mining-Programm parallel ausführen zu können, so dass jede Programminstanz eine Teilmenge aller zu durchsuchenden Repositorys

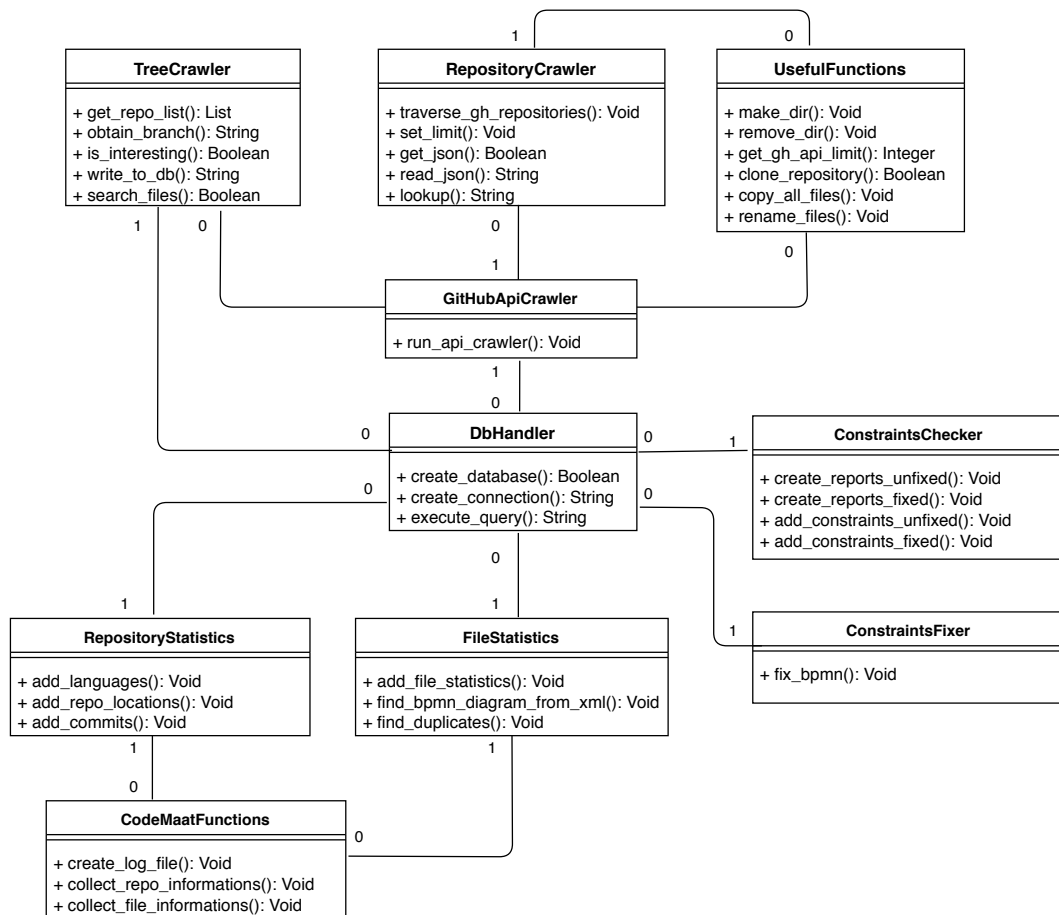


Abbildung 3.5.: Klassendiagramm des Mining-Programms

zugewiesen bekommt. Dann wird eine Liste der Namen und Zugangsdaten von Repositories aus der *to_query_projects* Datenbanktabelle ermittelt, die noch nicht durchsucht wurden. Für eine bestimmte Teilmenge aus dieser Liste werden die GitHub-Repositories nach BPMN-Prozessen durchsucht. Danach wird das *status* Attribut in der *to_query_projects* Tabelle auf 1 gesetzt, was bedeutet, dass diese Repositories schon durchsucht wurden. Schließlich wird dieser Vorgang für die nächste Teilmenge der Liste wiederholt. Die Funktion *run_api_crawler()* terminiert, wenn die Liste komplett bearbeitet wurde. Für die Erledigung der Einzelaufgaben benutzt *run_api_crawler()* Funktionen aus den **RepositoryCrawler**, **TreeCrawler** und **UsefulFunctions** Klassen.

Die **RepositoryCrawler** Klasse hat die zentrale Bedeutung im Programm. Wie im Abschnitt 3.1.3 beschrieben wurde, definiert die GitHub API 5000 Anfragen pro Stunde und Konto für authentifizierte Benutzer. Um diese Anzahl nicht zu überschreiten, prüft die Funktion *set_limit()* die Anzahl der verfügbaren GitHub API Anfragen und wenn sie verbraucht wurden, pausiert das Programm, bis neue 5000 Anfragen nach einer Stunde wieder zur Verfügung stehen. Andere Funktionen dieser Klasse (*traverse_gh_repositories()*, *get_json()*, *read_json()*, *lookup()*) erledigen den im Abschnitt 3.1.3.2 beschriebenen Vorgang, das heißt, für jedes GitHub-Repository wird seine Ordner- und Dateistruktur mithilfe der API ermittelt und in einer JSON-Datei gespeichert.

Die **TreeCrawler** Klasse enthält Funktionen, die es möglich machen, potentielle BPMN-Prozesse innerhalb des jeweiligen Repository zu finden. Die *get_repo_list()* Funktion ermittelt aus einem Ordner mit heruntergeladenen JSON-Dateien die Namen und Inhaber von Repositories und liefert sie in einer Liste zurück. Dann wird die *search_files()* Funktion aufgerufen, wobei für jedes Repository aus der Liste seine Baumstruktur traversiert wird und für jeden Dateinamen die *is_interesting()* Funktion aufgerufen wird. Letztere überprüft, ob sich in dem als Parameter übergebenen Dateinamen einschließlich der Dateiendung das gesuchte Schlüsselwort **bpmn** befindet und liefert im Erfolgsfall *Wahr* oder sonst *Falsch* zurück. Falls das Schlüsselwort **bpmn** ein Teil des Dateinamens ist, wird die Funktion *obtain_branch()* aufgerufen, um *branch*, indem die Datei sich befindet, zu ermitteln. Dies wird benötigt, um das Weblink für den gefundenen potentiellen BPMN-Prozess zu bilden. Nachdem das passiert ist, wird Funktion *write_to_db()* aufgerufen. In dieser werden die ermittelten Daten, das heißt Name des Repository, Zugangsdaten, Weblink und Dateipfad in der Datenbanktabelle *result_files* gespeichert.

Es ist wichtig, nachdem ein potentieller BPMN-Prozess gefunden wurde, diesen für die spätere Analyse zu sichern. Denn der Inhaber der Datei kann sie umbenennen, verschieben oder löschen, aufgrund dessen sie unter dem in der *result_files* Datenbanktabelle gespeicherten Link nicht mehr verfügbar wird. Dafür wird in der *search_files()* Funktion für jedes GitHub-Repository, in dem mindestens ein potentieller Prozess gefunden wurde, die *clone_repository()* Funktion aus der **UsefulFunctions** Klasse aufgerufen und das Repository mit dem Befehl *git clone* heruntergeladen.

Die Funktionen der **UsefulFunctions** Klasse dienen Zwecken, wie beispielsweise Erzeugen und Löschen eines Ordners oder Kopieren von Dateien. Die Funktionen werden aus anderen Klassen aufgerufen.

Die Klasse **DbHandler** spielt die Rolle des Vermittlers zwischen dem Programm und der Datenbank. Sie enthält Funktionen für das Erzeugen einer Datenbank: *create_database()*, für das Herstellen einer Verbindung zu einer Datenbank: *create_connection()* und für die Ausführung von Datenbankabfragen: *execute_query()*.

In der *execute_query()* Funktion wurde der Fall berücksichtigt, wenn mehrere Instanzen des Programms gleichzeitig versuchen in die selbe Tabelle zu schreiben. In diesem Fall wird nur ein Schreibvorgang zu gleichen Zeit durchgeführt. Versuchen andere Programminstanzen parallel einen Schreibvorgang, wird eine Ausnahme geworfen. Wenn das passiert, wird das betroffene Programm für 0.3 sec pausiert und danach ein neuer Versuch für den Schreibvorgang unternommen. Das wird wiederholt, bis der Schreibvorgang erfolgreich durchgeführt werden kann.

Nachdem alle Repositories aus der Datenbanktabelle **to_query_projects** durchsucht worden sind, befinden sich Informationen über gesammelte potentielle BPMN-Prozesse in der **result_files** Tabelle. Der nächste, zweite, Schritt ist somit, wie in Abbildung 3.4 zu sehen ist, die Ergebnisse nach BPMN-Prozessen im XML-Format zu filtern. Diese Aufgabe übernimmt die *find_bpmn_diagram_from_xml()* Funktion aus der **FileStatistics** Klasse. Es wird zuerst eine Liste von Textdateien

gebildet, die im XML-Format gespeichert sind. Danach wird in jeder Datei nach der „<http://www.omg.org/spec/BPMN/20100524/MODEL>“ Sequenz gesucht. Denn sie soll nach der Spezifikation von BPMN [16], als Wert des `xmlns` Attributs im einleitenden Tag stehen. Für jede XML-Datei, die diese Sequenz enthält, wird der Wert des Attributs `is_xml` in der Tabelle **result_files** auf 1 gesetzt.

Nach der Filterung beginnt somit der letzte, dritte, Schritt aus der Abbildung 3.4. Hier werden verschiedene Informationen sowohl über BPMN-Prozesse, als auch über Repositories, zu denen sie gehören, gesammelt. Dafür wurden im Mining-Programm vier externe Werkzeuge benutzt:

- Code Maat [18] - ein Programm, mit dem Daten aus Versionskontrollsystemen abgerufen und analysiert werden können [48]. Code Maat benutzt die log-Datei mit der Änderungshistorie eines Repository, um solche Metriken, wie Anzahl der Commits und Autoren oder Dateialter zu berechnen.
- Duplicate Files Finder [21] - eine Anwendung, die nach doppelten Dateien sucht (Dateien mit demselben Inhalt, aber nicht unbedingt demselben Namen) und dem Benutzer das Entfernen doppelter Dateien ermöglicht, indem sie entweder gelöscht oder Verknüpfungen erstellt werden [21]. Bei der Suche nach Duplikaten werden zuerst alle Dateien nach ihrer Größe sortiert und dann Dateien mit gleicher Größe miteinander verglichen [21].
- BPMNspector [15] - ein Programm, das einzelne Dateien oder vollständige Verzeichnisse auf BPMN-Dateien überprüft und Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln meldet [15].
- BPMNspector-fixSeqFlow [14] - ein Programm, das die BPMN-Prozesse mit dem Verstoß „EXT.023“ (siehe Tabelle C.1 im Anhang C.1) repariert, indem es „die fehlenden `<incoming>` / `<outgoing>` -Elemente hinzufügt und eine korrigierte Version des BPMN-Prozesses erstellt“ [14].

Die Klasse **CodeMaatFunctions** enthält drei Funktionen: `create_log_file()`, `collect_file_informations()` und `collect_repo_informations()`. Die `create_log_file()` Funktion ist dafür zuständig, für ein GitHub Repository die gesamte Historie seiner Commits zu bilden und in einer log-Datei zu speichern. In den `collect_file_informations()` und `collect_repo_informations()` Funktionen wird Programm Code Maat aufgerufen und die generierten log-Dateien als Eingabequelle benutzt.

Die `collect_file_informations()` Funktion erstellt für ein geklontes GitHub-Repository zwei csv-Dateien. In erster werden für jede Datei innerhalb des jeweiligen Repository die Anzahl der Autoren und Dateiänderungen gespeichert. In der zweiten Datei werden für jede Datei innerhalb des jeweiligen Repository ihr Alter in Monaten (Zeitraum seit letzter Änderung) gespeichert.

Die `collect_repo_informations()` Funktion erzeugt für ein GitHub-Repository zwei csv-Dateien. In erster csv-Datei wird gespeichert, an welchem Tag wie viel Dateien erzeugt und gelöscht wurden und wie viel Commits gemacht wurden. In zweiter csv-Datei wird die Anzahl von Commits seit der Dateierstellung und Anzahl der Autoren gespeichert.

Die **FileStatistics** Klasse enthält außer der schon beschriebenen Funktion *find_bpmn_diagram_from_xml()* auch noch die Funktionen *add_age_n_authors_n_revers()*, *count_process_elements()* und *find_duplicates()*. Die *add_age_n_authors_n_revers()* Funktion benutzt *create_log_file()* und *collect_file_informations()* als Unterrouinen und speichert in der *file_statistics* Datenbanktabelle für jede Datei, die sich dort befindet, das Dateialter in Monaten und die Anzahl der Autoren und Änderungen.

Vor der Benutzung der *find_duplicates()* Funktion soll die *copy_all_files* Funktion aus **UsefulFunctions** Klasse aufgerufen werden. Die letzte kopiert alle Dateien, dessen Name in der *result_files* Datenbanktabelle stehen, in einen Ordner. Die Dateikopien werden dabei umbenannt, denn es kann sein, dass manche Dateien aus unterschiedlichen Repositorys den gleichen Namen haben können. Der Name der Kopie wird im Attribut *path_copy_file* in der **result_files** Datenbanktabelle gespeichert.

Danach kann die *find_duplicates()* Funktion aufgerufen werden, die das Programm Duplicate Files Finder als Unteroutine nutzt. Duplicate Files Finder erstellt aus dem Ordner mit Dateikopien eine txt-Datei mit Dateinamen, die Duplikate haben. Diese Datei wird dann weiter in *find_duplicates()* Funktion analysiert und für jede zwei und mehr Dateien, deren Inhalte gleich sind, wird die gleiche Integer Zahl in der *duplicate* Spalte von **result_files** Datenbanktabelle gespeichert.

Die **RepositoryStatistics** Klasse enthält Funktionen, deren Aufgabe es ist, Informationen über ein geklontes GitHub-Repository zu gewinnen und in die **result_projects** Datenbanktabelle zu schreiben. Die *add_languages()* Funktion speichert für jedes GitHub-Repository die dominierende Programmiersprache, die mittels GitHub API abgefragt wird.

Die *add_repo_location()* Funktion berechnet für jedes GitHub-Repository Ländercodes für die Länder, in denen sich die am Repository Mitwirkenden befinden, falls die Information vorliegt. Das wird in zwei Schritten gemacht. In erstem Schritt wird für jedes Repository die Liste der Mitwirkenden ermittelt. Das wird mithilfe von HTTP-Anfragen an die GitHub API gemacht. Im zweiten Schritt wird für jeden Entwickler aus der Liste sein Standort ermittelt, indem diese Information aus der GHTorrent-Datenbanktabelle **users** entnommen und in der Datenbanktabelle **result_projects** gespeichert wird.

Die *add_commits()* Funktion benutzt als Unteroutine die *collect_repo_informations()* Funktion aus der **CodeMaatFunctions** Klasse und berechnet für ein geklontes GitHub-Repository das erste Commit, das letzte Commit und die Gesamtanzahl der Commits. Diese Informationen werden in der **result_projects** Datenbanktabelle gespeichert.

Die **ConstraintsChecker** Klasse wird verwendet, um die im XML-Format vorliegenden BPMN-Prozesse zu analysieren. Die Funktionen *create_reports_unfixed()* und *create_reports_fixed()* benutzen als Unteroutine BPMNspectator. Dieses erzeugt für jeden BPMN-Prozess einen Bericht in XML-Format, in dem steht, ob der Prozess Verstöße gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln hat und wenn das der Fall ist, werden sie aufgelistet.

Die `create_reports_unfixed()` Funktion wird vor der Anwendung des Programms BPMN spectator-fixSeqFlow aufgerufen und die `create_reports_fixed()` Funktion - nach der Anwendung dieses Programms. Die Funktionen `add_constraints_unfixed` und `add_constraints_fixed` fügen die mit oben genannten Funktionen gesammelten Daten in die Datenbanktabelle **result_xml_original** ein.

Die **ConstraintsFixer** Klasse enthält nur eine Funktion: `fix_bpmn()`. Diese Funktion benutzt als Unteroutine das Programm BPMN spectator-fixSeqFlow, mit dem BPMN-Prozesse mit dem Verstoß „EXT.023“ (siehe Tabelle C.1 im Anhang C.1) repariert werden.

Als Beispiel für einen Regelverstoß gegen die in der BPMN-Spezifikation definierten Syntax- und Semantikregeln betrachten wir einen der häufigsten Fehler in BPMN-Prozessen: eine Verletzung einer `<sequenceFlow>` Einschränkung (in BPMN spectator mit „EXT.023“ bezeichnet) [14]. Die Abbildung 3.6 zeigt uns einen einfachen BPMN-Prozess, der weiter in den Abbildungen 3.7 und 3.8 im XML-Format zu sehen ist. Der in der Abbildung 3.7 dargestellte Prozess verletzt die `<sequenceFlow>` Einschränkung, indem in den Definitionen vom Startereignis, Enderereignis und Task die ein- oder ausgehende Kante (Sequence Flow) nicht erwähnt wird. Die korrekte Version vom Prozess 3.7 ist in der Abbildung 3.8 zu sehen.



Abbildung 3.6.: Ein einfacher BPMN-Prozess

```

1 <process id="P1" name="A Basic Process" isExecutable="true">
2   <startEvent id="start" name="Start"></startEvent>
3
4   <sequenceFlow id="flow1" sourceRef="start" targetRef="task"></sequenceFlow>
5
6   <task id="task" name="Do The Work!"></task>
7
8   <sequenceFlow id="flow2" sourceRef="task" targetRef="end"></sequenceFlow>
9
10  <endEvent id="end" name="The End."></endEvent>
11 </process>
  
```

Abbildung 3.7.: Ein einfacher BPMN-Prozess mit Verletzung einer `<sequenceFlow>` Einschränkung. Übernommen aus [14].

```

1 <process id="P1" name="A Basic Process" isExecutable="true">
2   <startEvent id="start" name="Start">
3     <outgoing>flow1</outgoing>
4   </startEvent>
5
6   <sequenceFlow id="flow1" sourceRef="start" targetRef="task"></sequenceFlow>
7
8   <task id="task" name="Do The Work!">
9     <incoming>flow1</incoming>
10    <outgoing>flow2</outgoing>
11  </task>
12
13  <sequenceFlow id="flow2" sourceRef="task" targetRef="end"></sequenceFlow>
14
15  <endEvent id="end" name="The End.">
16    <incoming>flow2</incoming>
17  </endEvent>
18 </process>
  
```

Abbildung 3.8.: Ein einfacher BPMN-Prozess ohne Verletzung einer `<sequenceFlow>` Einschränkung. Übernommen aus [14].

In der *add_constraints()* Funktion werden die erzeugte Berichte analysiert. Für jeden Prozess wird geprüft ob er valide ist und wenn ja, das *valide* Attribut in der **result_bpmn** Datenbanktabelle auf 1 gesetzt, wenn nein - auf 0. Falls der Prozess nicht valide ist, d.h. BPMNspector einige Verstöße gefunden hat, werden die Namen der Verstöße und ihre Anzahl innerhalb des Prozesses zu einer Liste gebildet und in dem *constraints_list* Attribut in der **result_bpmn** Datenbanktabelle gespeichert.

4. Analyse gefundener BPMN-Dateien

In diesem Kapitel werden zuerst die Resultate der Ausführung des im vorangegangenen Kapitel beschriebene Mining-Programms vorgestellt. Danach werden die mit dem Mining-Pogramm gesammelten Daten analysiert. Hier werden zuerst allgemeine Statistiken über gefundene Dateien und ihre Repositories dargestellt und Statistiken über die Korrektheit der BPMN-Prozesse im XML-Format, die mit Hilfe des Analysewerkzeugs BPMNspector generiert wurden, präsentiert. Weiter werden BPMN-Prozesse im XML-Format, in denen BPMNspector Verstöße gegen BPMN 2.0-Einschränkungen gefunden hat, mit dem Reparaturwerkzeug BPMNspector-fixSeqFlow korrigiert und anschließend mit dem Programm BPMNspector wiederholt analysiert. Dieses Kapitel bildet somit die zwei letzten Schritte aus dem allgemeinen Algorithmus für das MSR, die Datenextraktion und Datenanalyse, ab (siehe Abb. 2.4).

4.1. Resultate des Mining-Programms

Es wurde der GHTorrent-Dump vom 1.11.2018 genutzt, in dem es 61.632.173 nicht-gelöschte und nicht-geforkte Repositories gibt. Damit der Mining-Prozess nicht länger als 1,5 Monate dauert, wurde die Entscheidung getroffen, nur 10 Prozent, also 6.163.217 Repositories zufällig auszuwählen und nach BPMN-Prozessen zu durchsuchen.

Das implementierte Mining-Programm wurde dazu auf 4 Rechnern im Linux-Pool 1 der Fakultät für Mathematik und Informatik an der Friedrich Schiller Universität Jena ausgeführt. Jedes Programm hatte die Aufgabe, circa 1.540.804 Repositories zu durchsuchen.

Die Arbeit des Programms hat auf einem der 4 Rechner durchschnittlich 30 Tage, 12 Stunden und 48 Minuten gedauert. Die aufsummierte Laufzeit von 4 Rechnern beträgt somit 122 Tage, 3 Stunden und 11 Minuten.

Diese Daten lassen zu, die mittlere Anzahl von Repositories, die pro Stunde auf einem der 4 Rechner durchsucht wurden, zu berechnen. Es wurden also pro Stunde circa 2.103 Repositories durchsucht.

Es muss erwähnt werden, dass während der Laufzeit der Programme einer der Rechner von anderen Poolnutzern versehentlich ausgeschaltet wurde. Die Programme wurden jedoch mittels SSH-Verbindung periodisch manuell geprüft, und falls nötig neu gestartet, so dass Unterbrechungen nicht länger als 3 Stunden andauerten.

Es wurden 1.251 Repositorys gefunden (von 6.163.217 durchsuchten Repositorys), die mindestens einen potentiellen BPMN-Prozess (es kommt „bpmn“ im Dateinamen vor) enthielten. Insgesamt wurden 21.306 potentielle BPMN-Prozesse gefunden.

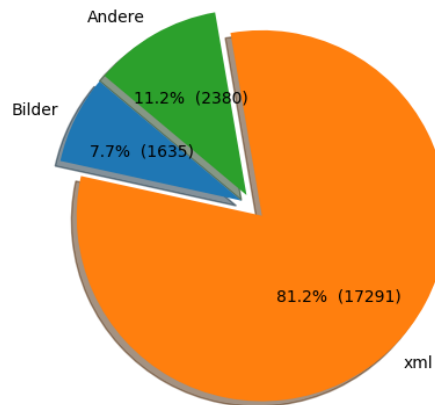


Abbildung 4.1.: Dateiformate von potentiellen BPMN-Prozessen (gruppiert)

Das Diagramm 4.1 zeigt eine Übersicht zu den dabei gefundenen Dateiformaten. Es ist zu sehen, dass die absolute Mehrheit (81,2%) der gefundenen Dateien das XML Format haben. Ihre Anzahl ist 17.291. Der Anteil an Bilddateien und Dokumenten, zu denen Dateien mit den Dateierendungen *.png, *.gif, *.jpeg, *.jpg, *.svg und *.pdf zählen, beträgt 7,7% oder 1.635 Dateien. Der Anteil aller anderen Dateiformate, beispielsweise Dateien mit den Dateierendungen *.js oder *.jar, beträgt 11,2% von allen gefundenen Dateien.

Die 17.291 Dateien im XML-Format wurden weiter analysiert. Als Ergebnis der Analyse, vergleiche auch den zweiten Schritt in Abbildung 3.4, ergab sich eine weitere Reduktion auf 16.907 XML-Dateien, die als BPMN-Prozesse erkannt wurden. Diese Dateien stammten aus 928 GitHub-Repositorys.

In der Tabelle 4.1 sind die Resultate des Mining-Programms dargestellt.

	Dateien	Repositorys
Insgesamt	21.306	1.251
BPMN-Prozesse im XML-Format	16.907	928

Tabelle 4.1.: Resultate des Mining-Programms

4.2. Allgemeine Statistiken

Wir haben die allgemeinen Statistiken in zwei Gruppen unterteilt: Statistiken über GitHub-Repositorys, die BPMN-Prozesse enthalten, und Statistiken über die gefundenen BPMN-Dateien selbst.

4.2.1. Statistiken über GitHub-Repositorys

Die Repositorys mit BPMN-Prozessen wurden hinsichtlich der dominanten Programmiersprache, dem Zeitpunkt der Erstellung (Alter), dem Zeitpunkt der letzten Änderung, der Gesamtdauer der Aktivität, Anzahl der Commits und dem Standort der Repositoryentwickler untersucht.

4.2.1.1. Dominante Programmiersprache

Diese Statistik wurde mittels GitHub API ermittelt, siehe Funktion `add_languages()` im Mining-Programm [40] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.2 zeigt die dominanten Programmiersprachen in den GitHub-Repositorys, die BPMN-Prozesse enthalten. Aus diesem Diagramm kann man ablesen, dass jedes zweite Repository (51,2%) Java als dominante Programmiersprache hat und mehr als jedes dritte Repository (37%) - JavaScript. Diesen Sprachen mit einem großen Abstand folgen die Beschreibungssprache HTML (7,1%) und andere Programmiersprachen (<2%).

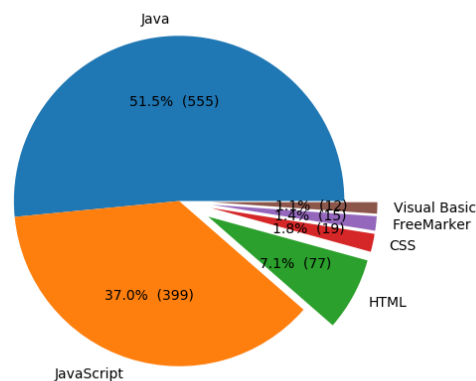


Abbildung 4.2.: Dominante Programmiersprachen der Repositorys mit BPMN-Dateien

Um festzustellen, ob es Unterschiede zwischen den oben genannten Statistiken über dominante Programmiersprachen von GitHub-Repositorys mit BPMN-Prozessen und allen GitHub-Repositorys gibt, schauen wir jetzt auf die Abbildung 4.3. Diese zeigt die dominanten Programmiersprachen von allen öffentlichen GitHub-Repositorys, erstellt im 1. Quartal 2019. Es ist zu sehen, dass JavaScript, Python und Java zu den dominanten Programmiersprachen auf GitHub gehören. Der Vergleich der Abbildungen 4.2 und 4.3 zeigt uns, dass es einige Ähnlichkeiten aber auch Unterschiede gibt. So gehören in beiden Abbildungen JavaScript und Java zu den zwei meistgenutzten Programmiersprachen. Jedoch bei Repositorys mit BPMN-Prozessen ist Java an erster Stelle und bei allen GitHub-Repositorys - JavaScript. Auch alle anderen Beschreibung- und Programmiersprachen aus der Abbildung 4.2, wie HTML, CSS, FreeMarker und Visual Basic gehören allgemein nicht zu den am häufigsten auftretenden Programmiersprachen auf GitHub.

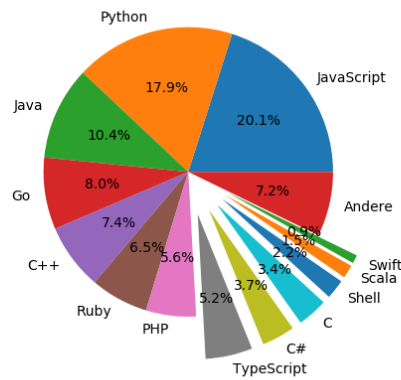


Abbildung 4.3.: Dominante Programmiersprachen von allen öffentlichen GitHub-Repositorys, erstellt im 1. Quartal 2019 [28]

4.2.1.2. Erstellungsjahre und Jahre der letzten Änderung

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_commits()` im Mining-Programm [40] und Beschreibung im Abschnitt 3.2.2.

Das Kreisdiagramm in der Abbildung 4.4 zeigt uns eine Dynamik der Erstellung von neuen GitHub-Repositorys mit BPMN-Prozessen. Es ist zu sehen, dass zwischen 2005 und 2010 genau so viele Repositorys erstellt wurden, wie in 2011. Nach 2011 wurde diese Anzahl jedes Jahr deutlich höher, als im Jahr davor. Der Grund dafür besteht vermutlich darin, dass im Januar 2011 die neue Version 2.0 von BPMN erschien. Seitdem wuchs die Anzahl der neu-erstellten Repositorys stetig und zum Beispiel wurden in 2017 fast 5-Mal so viele Repositorys mit BPMN-Prozessen erstellt, als drei Jahre davor in 2014. Im Durchschnitt betrug der Zuwachs der GitHub-Repositorys mit BPMN-Prozessen zwischen 2011 und 2017 ca. 71.7% pro Jahr. Die Daten für 2018 sind nicht komplett. Das liegt daran, dass wir den GHTorren-Datensatz von 01.11.2018 benutzt haben, der Informationen nur bis zu diesem Datum enthielt.

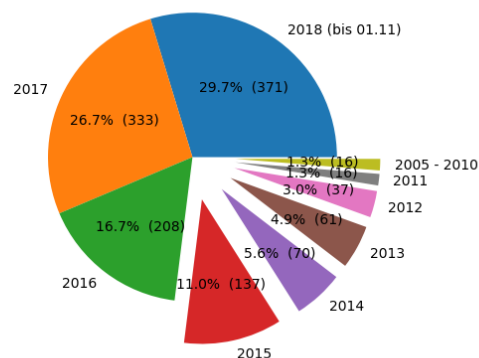


Abbildung 4.4.: Jahre, in denen Repositorys mit BPMN-Dateien erstellt wurden

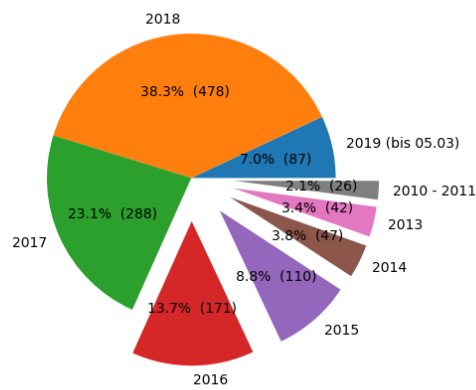


Abbildung 4.5.: Jahre, in denen Repositorys mit BPMN-Dateien letztes Mal geändert wurden

Das Diagramm in der Abbildung 4.5 liefert uns Informationen darüber, wann GitHub-Repositorys mit BPMN-Prozessen das letzte Mal geändert wurden. Dieses Diagramm zeigt, dass im Unterschied zur Abbildung 4.4, wo die ersten Repositorys im Jahre 2005 erstellt wurden, die ersten Repositorys im Jahre 2010 abgeschlossen wurden. D.h. alle zwischen 2005 und 2010 erstellten Repositorys waren mindestens bis 2010 aktiv. Aus dem Diagramm kann man auch entnehmen, dass die Anzahl der zum letzten Mal geänderten Repositorys jedes Jahr zwischen 2010 und 2018 immer größer war als im Jahr davor. Die Daten für das Jahr 2019 sind nicht komplett - nur bis 5. März. Das liegt daran, dass das Klonen der GitHub-Repositorys am 05.03.2019 abgeschlossen wurde und wir Informationen über GitHub-Repositorys nur bis zu diesem Datum hatten.

Um die Abbildungen 4.4 und 4.5 zu vergleichen, wurde aus beiden Grafiken ein gemeinsames Diagramm 4.6 mit absoluter Häufigkeit erstellt und zwar für die Jahre zwischen 2010 und 2017, für die es komplette Informationen auf beiden Grafiken gab. Das Diagramm 4.6 zeigt uns deutlich, dass die Anzahl der neu-erstellten Repositorys mit BPMN-Prozessen in einem der betrachteten Jahren größer war, als die Anzahl der Repositorys, die in jeweiligem Jahr zum letzten Mal geändert wurden. Daraus kann man schließen, dass in diesen Jahren mehr öffentlichen GitHub-Repositorys mit BPMN-Prozessen erstellt, als abgeschlossen wurden.

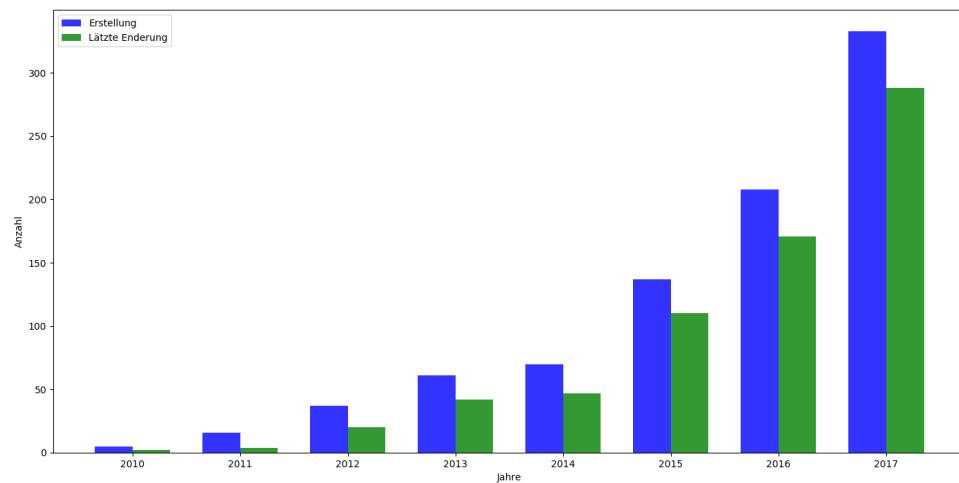


Abbildung 4.6.: Jahre, in denen Repositorys mit BPMN-Dateien erstellt und letztes Mal geändert wurden

4.2.1.3. Dauer der Aktivität

Die Abbildung 4.7 zeigen Informationen über die Dauer der Aktivität (Zeitraum zwischen Erstellung und letzter Änderung) von Repositorys mit BPMN-Dateien. Es ist zu sehen, dass 29,1% (364) der Repositorys nur innerhalb eines Tages aktiv waren und 25,1% (314) der Repositorys mehr als ein Tag und weniger als ein Monat aktiv waren. Das heißt, es waren insgesamt mehr als die Hälfte der Repositorys (54,2%) weniger als ein Monat aktiv. Fast 20% (248) der Repositorys waren in einem Zeitraum von 1 bis 6 Monaten aktiv. Signifikant ist, dass es 48 Repositorys gibt, die mehr als 5 Jahre aktiv waren.

Die durchschnittliche Dauer der Aktivität von allen Repositorys mit BPMN-Dateien beträgt ca. 235 Tage, was 7 Monaten und 17 Stunden entspricht.

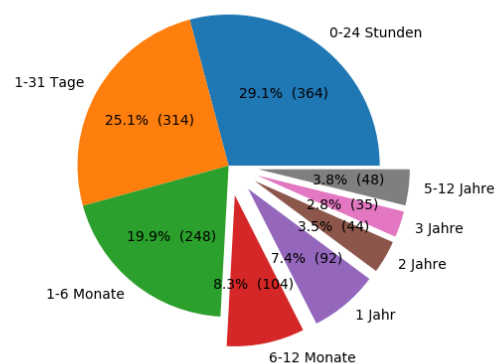


Abbildung 4.7.: Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien

4.2.1.4. Anzahl der Commits

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_commits()` im Mining-Programm [40] und Beschreibung im Abschnitt 3.2.2.

Das nächste Merkmal von Repositorys mit BPMN-Dateien, das untersucht wurde, war die Anzahl der Commits, die seit der Repositoryerstellung gemacht wurden. Aus der Abbildung 4.8 kann man entnehmen, dass 12,5% der Repositorys nur ein Commit hat und 24,1% der Repositorys von 2 bis 5 Commits haben. Insgesamt kann man aus diesem Diagramm entnehmen, dass ca. die Hälfte der Repositorys (51,7%) 10 oder weniger Commits haben. Jedes vierte Repository (26,5%) hat zwischen 11 und 50 Commits. 5,5% der Repositorys haben mehr als 500 Commits. Die größte Anzahl der Commits, die ein Repository hat, beträgt 10.056.

Die durchschnittliche Anzahl der Commits von allen Repositorys mit BPMN-Dateien beträgt fast 171 Commits.

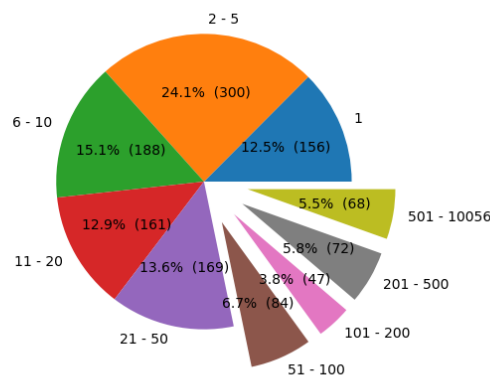


Abbildung 4.8.: Anzahl der Commits von GitHub-Repositorys mit BPMN-Dateien

4.2.1.5. Standorte der Repository-Entwickler

Diese Statistik wurde mittels GitHub API und Daten aus dem GHTorrent-Projekt ermittelt, siehe Funktion `add_repo_location()` im Mining-Programm [40] und Beschreibung im Abschnitt 3.2.2.

Ein weiteres Merkmal zu den Repositorys, das untersucht wurde, war der Standort der Repository-Entwickler. Die Ergebnisse sind in der Abbildung 4.9 dargestellt. Diese Grafik wurde jedoch nur anhand der Standorte von 395 GitHub-Repositorys mit BPMN-Prozessen gebildet. Das liegt daran, dass die Datenbanktabelle „users“ (Abbildung 3.3), deren Inhalt aus dem Datensatz des GHTorrent-Projekts stammt und die Informationen über alle GitHub-Benutzer enthält, für viele von ihnen im Feld „country_code“ keine Angabe hat.

Beim Betrachten der Abbildung 4.9 fällt sofort auf, dass die meisten Entwickler, die zu den GitHub-Repositorys mit BPMN-Prozessen ihr Code schrieben, aus China (14,7%), Deutschland (14,4%) und aus den USA (14,4%) stammen. Ihnen mit einem

großen Abstand folgen Programmierer aus der Schweiz (3,8%), Frankreich (3,5%) und anderen Ländern.

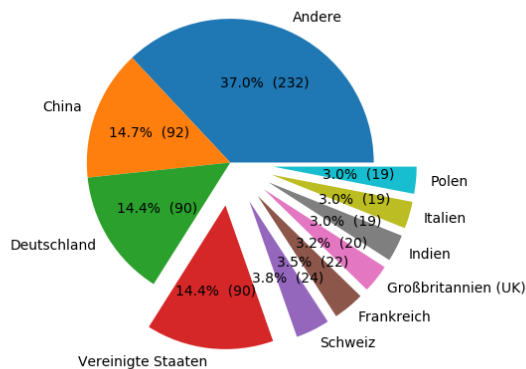


Abbildung 4.9.: Standorte der Repository-Entwickler

Die Abbildung 4.10 enthält die Länder, aus denen die meisten GitHub-Benutzer kommen. Diese Abbildung ähnelt der Abbildung 4.9 insofern, dass in beiden USA und China die zwei Länder mit den meisten GitHub-Benutzern sind. Aber im Gegensatz zur Abbildung 4.9, wo Deutschland und USA den zweiten Platz teilen, befindet sich dieses europäische Land in der Abbildung 4.10 auf dem 5. Platz. Insgesamt kommen in beiden Abbildungen sechs dieselbe Länder vor.

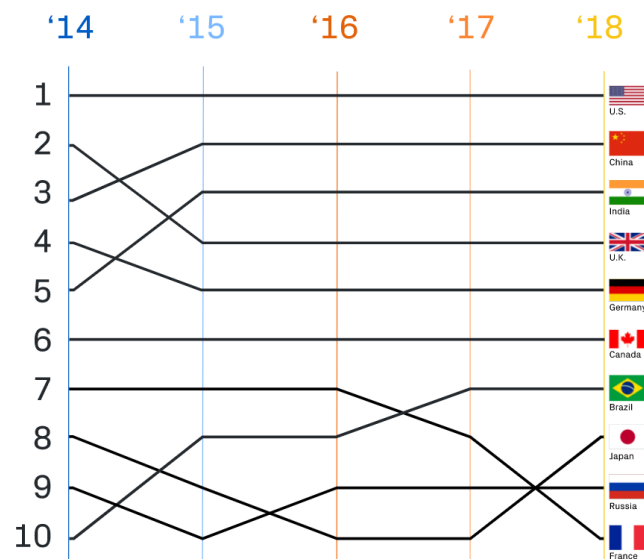


Abbildung 4.10.: Standorte mit den meisten GitHub-Benutzer [38]

4.2.2. Statistiken über gefundene BPMN-Dateien

4.2.2.1. Analyse der Dateien bezüglich Duplikate

Diese Statistik wurde mittels des Programms Duplicate Files Finder ermittelt, siehe Funktion `find_duplicates()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

In der Abbildung 4.18 ist zu sehen, dass nur für 37,7% der BPMN-Dateien keine Duplikate gefunden wurden. Der Anteil der Dateien, die mindestens einmal dupliziert wurden, das heißt mehrfach in einem oder mehreren Repositorys vorkamen, beträgt 12,6% und der Anteil der reinen Duplikate (Kopien von anderen BPMN-Dateien) 49,7%.

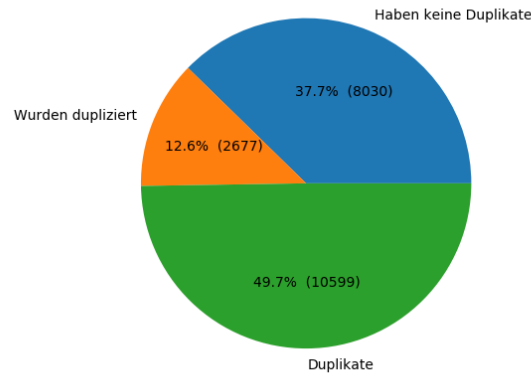


Abbildung 4.11.: Anteil BPMN-Dateien, die dupliziert wurden

In Abbildung 4.12 sind Informationen zu den identifizierten 10.599 Duplikaten (Kopien von anderen BPMN-Dateien) dargestellt. Es ist zu sehen, dass 852 Dateien (31,8% der Duplikate) einmal dupliziert wurden und 571 Dateien (21,3% der Duplikate) zweimal dupliziert wurden. Interessant ist, dass es 2 Dateien gibt, die 170-Mal dupliziert wurden.

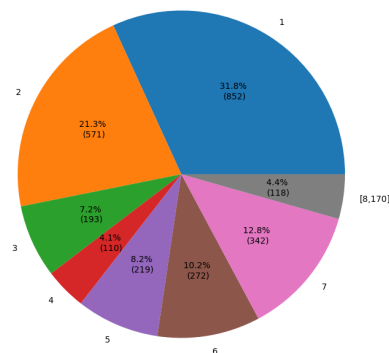


Abbildung 4.12.: Statistik über Duplikate von BPMN-Dateien

Die 4 am meisten duplizierten BPMN-Dateien sind im Anhang A.1 zu sehen.

Die gefundenen Duplikate (10.599 Dateien), wurden bei Statistik-Analysen über die grafische Repräsentation der BPMN-Prozesse im XML-Format (Abschnitt 4.2.2.2), Größe der BPMN-Prozesse im XML-Format (Abschnitt 4.2.2.3) und Statistik über die Korrektheit der BPMN-Prozesse im XML-Format (Abschnitt 4.3) nicht mitgezählt.

4.2.2.2. Grafische Repräsentation der BPMN-Prozesse im XML-Format

Diese Statistik wurde mittels eines Python-Skripts ermittelt, siehe Funktion `count_process_elements()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

Ein BPMN-Prozess im XML-Format, der der BPMN-Spezifikation in der Version 2.0 oder höher entspricht, enthält ein XML-Wurzelement `definitions`, das wiederum aus mehreren XML-Kinderelementen bestehen kann. Das XML-Element `definitions` enthält Deklarationen, die den Namensraum der XML-Datei bestimmen.

Die Abbildung 4.13 zeigt die XML-Repräsentation eines BPMN-Prozesses aus dem Abschnitt 3.2.2 (siehe Abbildung 3.6). An diesem Beispiel kann man sehen, dass das Wurzel-Element `definitions` (in der Abb. 4.13 `bpmn:definitions`) aus zwei Kinder-Elementen `process` (in der Abb. 4.13 `bpmn:process`) und `BPMNDiagram` (in der Abb. 4.13 `bpmn:BPMNDiagram`) besteht. Das XML-Element `process` definiert alle Elemente von dem BPMN-Prozess und das XML-Element `BPMNDiagram` definiert die grafische Anordnung der BPMN-Prozesselemente. Da das XML-Element `BPMNDiagram` nur die grafische Repräsentation eines Prozesses bestimmt, wird es oft in BPMN-Prozessen im XML-Format weggelassen.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpmn:definitions xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
3   xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
4   xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
5   xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
6   id="Definitions_1yqvvyxc" targetNamespace="http://bpmn.io/schema/bpmn"
7   exporter="Camunda Modeler" exporterVersion="2.0.3">
8   <bpmn:process id="Process_1" name="A Basic Process" isExecutable="true">
9     <bpmn:startEvent id="StartEvent_1" name="Start"></bpmn:startEvent>
10    <bpmn:task id="Task_1ve7ppt" name="Do The Work!"></bpmn:task>
11    <bpmn:sequenceFlow id="SequenceFlow_0kubrj1" sourceRef="StartEvent_1" targetRef="Task_1ve7ppt" />
12    <bpmn:endEvent id="EndEvent_0d529fy" name="The End."></bpmn:endEvent>
13    <bpmn:sequenceFlow id="SequenceFlow_1parnc1" sourceRef="Task_1ve7ppt" targetRef="EndEvent_0d529fy" />
14  </bpmn:process>
15  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
16    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_1">
17      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2" bpmnElement="StartEvent_1">
18        <dc:Bounds x="173" y="102" width="36" height="36" />
19        <bpmndi:BPMNLabel>
20          <dc:Bounds x="178" y="146" width="25" height="14" />
21        </bpmndi:BPMNLabel>
22      </bpmndi:BPMNShape>
23      <bpmndi:BPMNShape id="Task_1ve7ppt_di" bpmnElement="Task_1ve7ppt">
24        <dc:Bounds x="278" y="80" width="100" height="80" />
25      </bpmndi:BPMNShape>
26      <bpmndi:BPMNEdge id="SequenceFlow_0kubrj1_di" bpmnElement="SequenceFlow_0kubrj1">
27        <di:waypoint x="209" y="120" />
28        <di:waypoint x="278" y="120" />
29      </bpmndi:BPMNEdge>
30      <bpmndi:BPMNShape id="EndEvent_0d529fy_di" bpmnElement="EndEvent_0d529fy">
31        <dc:Bounds x="477" y="102" width="36" height="36" />
32        <bpmndi:BPMNLabel>
33          <dc:Bounds x="472" y="144" width="45" height="14" />
34        </bpmndi:BPMNLabel>
35      </bpmndi:BPMNShape>
36      <bpmndi:BPMNEdge id="SequenceFlow_1parnc1_di" bpmnElement="SequenceFlow_1parnc1">
37        <di:waypoint x="378" y="120" />
38        <di:waypoint x="477" y="120" />
39      </bpmndi:BPMNEdge>
40    </bpmndi:BPMNPlane>
41  </bpmndi:BPMNDiagram>
42 </bpmn:definitions>

```

Abbildung 4.13.: Beispiel eines BPMN-Prozesses im XML-Format

Die Abbildung 4.14 zeigt uns eine Statistik über das XML-Element `BPMNDiagram` der gefundenen BPMN-Prozesse im XML-Format. In die Analyse wurden 8.904 Dateien von den insgesamt 16.907 gefundenen BPMN-Dateien im XML-Format einbezogen. Bei den restlichen 8.003 Dateien handelt es sich um Kopien dieser Dateien. Es ist zu sehen, dass nur 63,8% (5.679) der Dateien das XML-Element

BPMNDiagram haben und bei 35% (3.118) der Dateien dieses Element fehlt. Bei 107 Dateien konnte nicht ermittelt werden, ob das XML-Element BPMNDiagram vorhanden ist.

Das bedeutet, dass nur 5.679 Dateien, die das XML-Element BPMNDiagram haben, mit einem BPMN-Editor, z.B. Camunda Modeler, grafisch dargestellt werden können. Damit 3.118 Dateien ohne das XML-Element BPMNDiagram grafisch darstellbar sind, muss in sie dieses fehlende XML-Element hinzugefügt werden. Das kann zum Beispiel mit dem Werkzeug bpmn-moddle-auto-layout [6] gemacht werden.

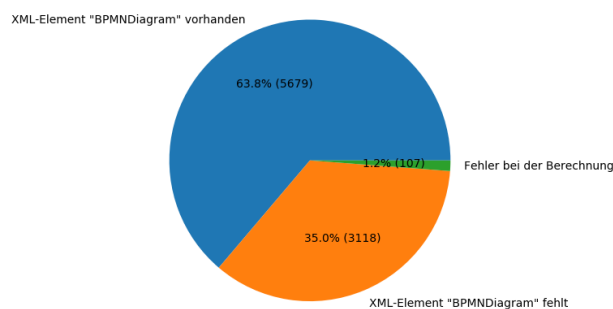


Abbildung 4.14.: Anteil der BPMN-Prozesse im XML-Format mit und ohne das XML-Element BPMNDiagram

4.2.2.3. Größe der BPMN-Prozesse im XML-Format

Diese Statistik wurde mittels eines Python-Skripts ermittelt, siehe Funktion `count_process_elements()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

Ein weiteres interessantes Merkmal, was untersucht wurde, war die Berechnung der Größe von BPMN-Prozessen im XML-Format. Als Maß für die Größe eines BPMN-Prozesses wurde die Anzahl der Kindknoten vom XML-Element `process` genommen. Das XML-Element BPMNDiagram wurde bei der Berechnung ignoriert, weil es in 35% der Dateien im XML-Format nicht vorhanden ist.

In die Analyse wurden 8.904 BPMN-Prozesse von den insgesamt 16.907 gefundenen BPMN-Prozessen im XML-Format einbezogen. Bei den restlichen 8.003 BPMN-Prozessen handelt es sich um Kopien dieser Dateien. Für 8.798 Dateien konnte die Anzahl der XML-Elemente berechnet werden und für die restlichen 107 Dateien, wegen einiger Ausnahmen bei der Berechnung, konnte es nicht gemacht werden.

Die Abbildung 4.15 zeigt uns die Resultate dieser Analyse. Aus dem Diagramm kann man entnehmen, dass 21,1% (1.881) der BPMN-Prozesse zwischen 1 und 10 XML-Elemente und 30,5% (2.714) der BPMN-Prozesse zwischen 11 und 20 XML-Elemente haben. Folglich haben fast die Hälfte der BPMN-Prozesse (48,4%) mehr als 20 XML-Elemente. Interessant ist, dass 0,6% (57) der BPMN-Prozesse mehr als 1001 XML-Elemente haben. Die größte Anzahl der XML-Elemente, die ein gefun-

dener BPMN-Prozess hat, beträgt 4.096.

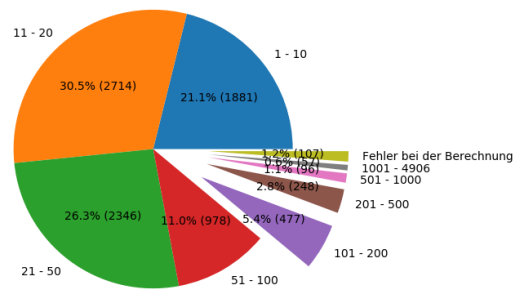


Abbildung 4.15.: Größe der BPMN-Prozesse im XML-Format

Die 5 größten BPMN-Prozesse im XML-Format, die das XML-Element BPMNDiagram haben und somit grafisch darstellbar sind, sind im Anhang B.1 zu sehen.

4.2.2.4. Autoren

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.16 zeigt uns die Anzahl der Autoren von BPMN-Dateien. Aus diesem Diagramm kann man ablesen, dass der überwiegende Anteil von allen BPMN-Dateien (92.1%) nur einen Autor hat. 4,6% der BPMN-Dateien haben zwei Autoren und 2,8% - drei. Mehr als 2

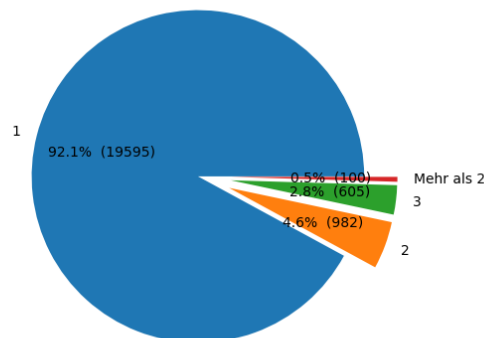


Abbildung 4.16.: Anzahl der Autoren von BPMN-Dateien

4.2.2.5. Dateialter

Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

Die Abbildung 4.17 zeigt uns das Alter von BPMN-Dateien (Zeitraum seit letzter Änderung). Es ist zu sehen, dass die Mehrheit der Dateien (36,1%) nicht älter als 1 Jahr ist. Fast jede vierte Datei ist 1 Jahr alt und jede sechste Datei - 2 Jahre alt. Die älteste BPMN-Datei wurde vor 11 Jahren erstellt. Im Ganzen kann man ablesen, dass seit 5 Jahren jedes Jahr mehr neue BPMN-Dateien erstellt werden, als im Jahr davor.

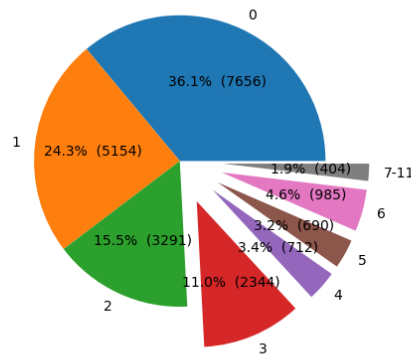


Abbildung 4.17.: Alter in Jahren von BPMN-Dateien

4.2.2.6. Dateiänderungen

Das nächste Merkmal von BPMN-Dateien, das untersucht wurde, war die Anzahl der Dateiänderungen inklusive der Dateierstellung. Diese Statistik wurde mittels des Programms Code Maat ermittelt, siehe Funktion `add_age_n_authors_n_revers()` im Mining-Programm [23] und Beschreibung im Abschnitt 3.2.2.

Wie aus der Abbildung 4.18 zu entnehmen ist, wurden drei Viertel von allen Dateien nach der Erstellung nicht mehr geändert. Der Anteil der Dateien, die zweimal geändert wurden, beträgt 15,9%. Der Anteil der Dateien, die mehr als 2-Mal geändert wurden, beträgt zusammen fast 8%.

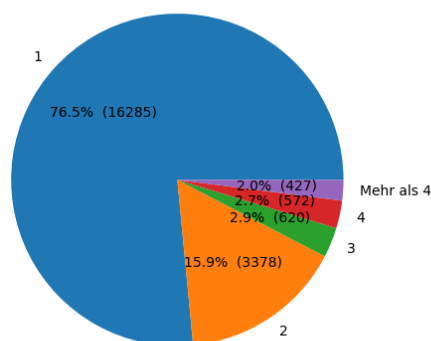


Abbildung 4.18.: Anzahl der Änderungen von BPMN-Dateien

4.3. Statistiken über Korrektheit der BPMN-Prozesse im XML-Format

4.3.1. Analyse mit dem Programm BPMNspector

Wie Matthias Geiger, der Autor von BPMNspector, angibt, werden BPMN-Prozesse häufig mit einigen Verstößen gegen die Syntax- und Semantikregeln von BPMN 2.0 erstellt [15]. Deshalb war es für uns von Interesse, die gefundenen BPMN-Dateien im XML-Format (BPMN-Prozesse) unter Verwendung des Analysewerkzeugs BPMNspector hinsichtlich solcher Verstöße zu untersuchen. Dieses Programm analysiert ein BPMN-Prozess oder ein Verzeichnis mit mehreren BPMN-Prozessen und ermittelt ob BPMN-Prozesse valide sind oder nicht. Falls ein Prozess nicht valide ist, liefert BPMNspector alle Verstöße gegen BPMN 2.0-Einschränkungen, die er hat. Die Liste von allen möglichen Verstößen ist in der Publikation „BPMN 2.0 Process Model Serialization Constraints“ von Matthias Geiger definiert (siehe [24]).

In die Analyse mit BPMNspector wurden 8.904 Dateien von den insgesamt 16.907 gefundenen BPMN-Dateien im XML-Format einbezogen. Bei den restlichen 8.003 Dateien handelt es sich um Kopien dieser Dateien. Siehe Funktionen `create_reports_unfixed()` und `add_constraints_unfixed()` im Mining-Programm [19] und Beschreibung im Abschnitt 3.2.2.

Wie in der Abbildung 4.19 zu sehen ist, wurden nur 16,5% (1.471) von den untersuchten BPMN-Prozessen als valide Prozesse, das heißt Prozesse ohne Verletzung von Syntax- oder Semantikregeln von BPMN 2.0, eingestuft. In 82,8% (7.376) der untersuchten BPMN-Prozesse wurden verschiedene Verstöße gegen BPMN 2.0-Einschränkungen gefunden. Die restlichen 0,6% (57) BPMN-Prozesse konnten mit dem Programm BPMNspector nicht analysiert werden.

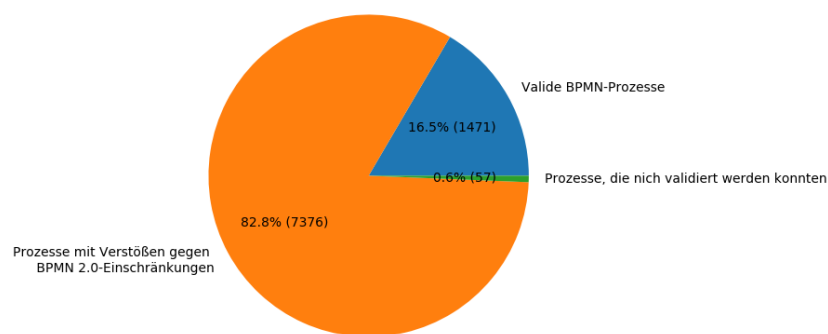


Abbildung 4.19.: Anteil valider und nicht-valider BPMN-Prozesse

Die Abbildung 4.20 zeigt uns eine Statistik zu 7.376 BPMN-Prozessen mit Verstößen gegen BPMN 2.0-Einschränkungen. In dieser Statistik wurden die 10 häufigsten Verstöße ermittelt, wobei das mehrmalige Vorkommen einer Art der Verstöße pro BPMN-Prozess nur einmal gezählt wurde. Es ist zu sehen, dass drei Arten der Verstöße, nämlich „EXT.101“, „EXT.023“ und „EXT.107“ am häufigsten gefunden wurden.

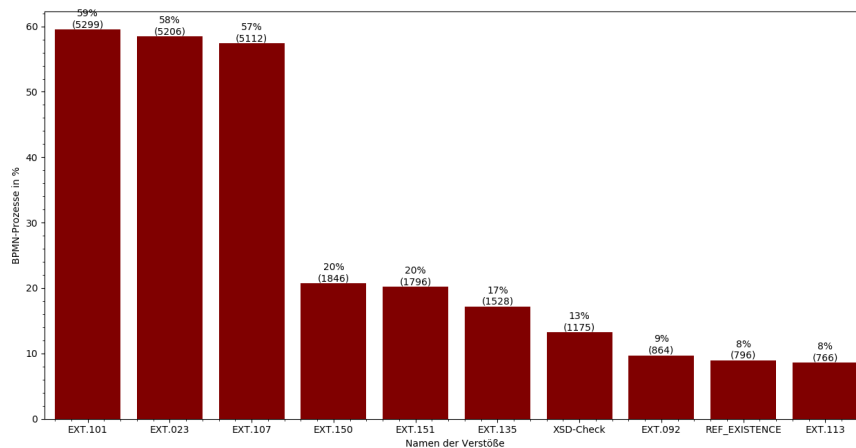


Abbildung 4.20.: BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen (insgesamt 7.376 BPMN-Prozesse)

Der „EXT.101“ Verstoß betrifft das Element Startereignis (StartEvent) eines BPMN-Prozesses. Hier geht es um eine Verletzung der Regel, die besagt, dass „ein Startereignis eine Quelle für einen Sequenzfluss sein muss“ [24].

Der Verstoß „EXT.023“ betrifft den Sequenzfluss (SequenceFlow) in BPMN-Prozessen. Es wird eine Regel verletzt, die besagt, dass „das Quell- und Zielelement des Sequenzflusses unter Verwendung der eingehenden oder ausgehenden Attribute auf die SequenceFlow-Definition verweisen muss“ [24] (siehe dazu auch die Diskussion in Abschnitt 3.2.2 und das Beispiel in Abbildungen 3.7 und 3.8). Wie der Autor von BPMNspector angibt, handelt es sich bei „EXT.023“ um einen der häufigsten Verstöße gegen die BPMN 2.0-Einschränkungen [14]. Unsere Analyse hat diese Aussage vollständig bestätigt. Matthias Geiger verweist auf das von ihm ebenfalls entwickelte Programm BPMNspector-fixSeqFlow [14], das diese Art von Verstößen korrigiert (siehe Anwendung von BPMNspector-fixSeqFlow im Abschnitt 4.3.2).

Der Verstoß „EXT.107“ betrifft das Element Endereignis (EndEvent). Es geht hier um Verletzung einer Regel, die besagt, dass „ein Endereignis mindestens einen eingehenden Sequenzfluss haben muss“ [24].

Die Bedeutungen von anderen Verstößen können aus der Tabelle C.1 im Anhang C.1 entnommen werden.

Die Abbildung 4.21 zeigt uns die Top-10 Verstöße gegen BPMN 2.0-Einschränkungen, aufsummiert über alle 7.376 BPMN-Dateien mit Verstößen. Es ist zu sehen, dass der Verstoß „EXT.023“ die größte Anzahl hat, nämlich 58.015 (38,6%). Im Unterschied zur Abbildung 4.20, wo „XSD-Check“ auf dem 7. Platz steht und in 13% der nicht validen BPMN-Prozessen vorkommt, befindet er sich bei der absoluten Zählung mit der Anzahl 43.516 (29%) auf dem zweiten Platz in der Abbildung 4.21. Prozentsätze der anderen Verstöße, die in der Abbildung 4.21 zu sehen sind, sind im Vergleich zu den schon erwähnten deutlich kleiner.

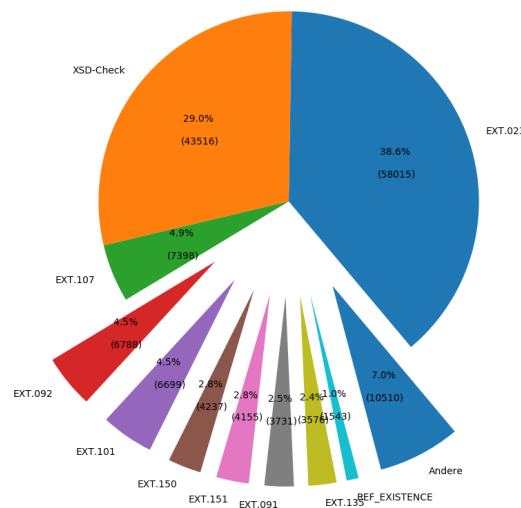


Abbildung 4.21.: Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 150.168 Verstöße)

Insgesamt hat die Anwendung von BPMNspector auf die gefundenen BPMN-Prozesse im XML-Format gezeigt, dass nur jeder sechste von ihnen in Hinsicht auf die BPMN-Spezifikation ein valider Prozess ist. Daraus kann man schließen, dass ein Bedarf für die Verbesserung der bestehenden BPMN-Werkzeuge existiert, um solche Fehler bei der Modellierung erkennen und beheben zu können. Die Grafiken 4.20 und 4.21 machen deutlich, welche Verstöße gegen BPMN 2.0-Einschränkungen bei der Erstellung der Prozesse berücksichtigt werden müssen.

4.3.2. Reparieren mit dem Programm BPMNspector-fixSeqFlow

Um die mit dem BPMNspector gefundenen Verstöße gegen BPMN 2.0-Einschränkungen zu reparieren, wurde das BPMNspector-fixSeqFlow Programm angewendet. Siehe Funktionen `fix_bpmn()` und `add_constraints_fixed()` im Mining-Programm [20], [19] und Beschreibung im Abschnitt 3.2.2. Wie in der Programmbeschreibung steht, repariert BPMNspector-fixSeqFlow den Verstoß „EXT.023“, indem es „die fehlenden `<incoming>` / `<outgoing>` -Elemente hinzufügt und eine korrigierte Version eines bestimmten BPMN-Prozesses erstellt“ [14].

Von 8.904 BPMN-Prozesse, für die das Programm BPMNspector-fixSeqFlow ausgeführt wurde, wurden 5.207 geändert. Diese 5.207 BPMN-Prozesse wurden wieder mit dem Programm BPMNspector analysiert. Die Analyse ergab, dass 3.270 BPMN-Prozesse keine Verstöße gegen die Syntax- und Semantikregeln von BPMN 2.0 mehr haben. Das bedeutet, dass sie mit dem BPMNspector-fixSeqFlow repariert wurden. In den restlichen 1.937 BPMN-Prozessen hat BPMNspector noch verschiedene Verstöße gefunden.

Der Vergleich der Verstöße vor und nach der Ausführung von BPMNspector-fixSeqFlow hat ergeben, dass 85.516 Verstöße repariert wurden. In der Tabelle 4.2 sind alle reparierten Verstöße aufgelistet. Es ist zu sehen, dass der größte Anteil der Verstöße, die repariert wurden, der Verstoß „EXT.023“ mit 67,5% ist. Ihm fol-

gen „EXT.107“ (8,5%), „EXT.101“ (7,6%) und andere reparierte Verstöße.

	Absolute Häufigkeit	Relative Häufigkeit	Verstoß
1	57.695	67,47%	„EXT.023“
2	7.224	8,45%	„EXT.107“
3	6.477	7,57%	„EXT.101“
4	4.025	4,71%	„EXT.150“
5	4.001	4,68%	„EXT.151“
6	3.295	3,85%	„EXT.135“
7	855	1,00%	„EXT.113“
8	630	0,74%	„EXT.116“
9	623	0,73%	„EXT.115“
10	605	0,71%	„EXT.025“
11	44	0,05%	„EXT.136“
12	20	0,02%	„EXT.063“
13	19	0,02%	„REF_EXISTENCE“
14	1	0,00%	„REF_TYPE“
15	1	0,00%	„EXT.084“
16	1	0,00%	„EXT.012“

Tabelle 4.2.: Reparierter Verstöße gegen BPMN 2.0-Einschränkungen

Die Bedeutungen von allen in der Tabelle 4.2 aufgelisteten Verstößen können aus der Tabelle C.1 im Anhang entnommen werden.

Im ganzen kann man schließen, dass das Hinzufügen der fehlenden `<incoming>` / `<outgoing>` -Elemente, was das Reparaturwerkzeug BPMNspector-fixSeqFlow tut, nicht nur Verstoß „EXT.023“, sondern auch andere Verstöße repariert, die in der Tabelle 4.2 zu sehen sind.

Die Abbildung 4.22 zeigt die Anteile valider und nicht-valider BPMN-Prozesse nach der Anwendung von BPMNspector-fixSeqFlow. Es ist zu sehen, dass der Anteil der reparierten BPMN-Prozesse im XML-Format 36,7% (3.270) beträgt. Im Ganzen beträgt der Anteil der validen BPMN-Prozesse nach der Anwendung von BPMNspector-fixSeqFlow mehr als die Hälfte, und zwar 53,2% (4.741). In 46,1% (4.106) der BPMN-Prozesse wurden weiterhin verschiedene Verstöße gegen BPMN 2.0-Einschränkungen gefunden. Die restlichen 0,6% (57) BPMN-Prozesse konnten mit dem Programm BPMNspector nicht analysiert werden.

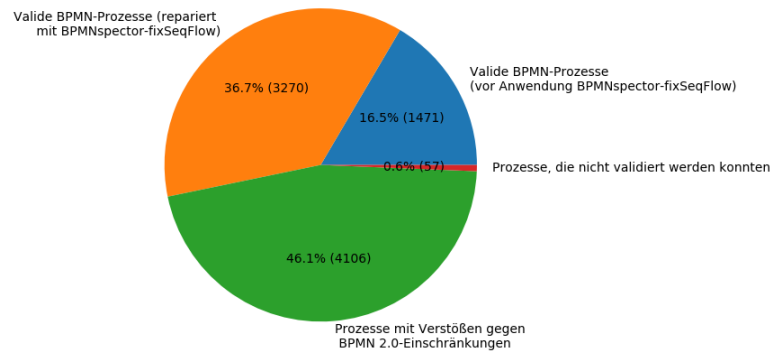


Abbildung 4.22.: Anteil valider und nicht-valider BPMN-Prozesse nach der Anwendung BPMNspector-fixSeqFlow

Die Abbildung 4.23 zeigt uns eine Statistik zu 4.106 BPMN-Prozessen mit Verstößen gegen BPMN 2.0-Einschränkungen nach der Anwendung von BPMNspector-fixSeqFlow. In dieser Statistik wurden die 10 häufigsten Verstöße ermittelt, wobei das mehrmalige Vorkommen einer Art der Verstöße pro BPMN-Prozess nur einmal gezählt wurde. Bei betrachten dieser Tabelle fällt sofort auf, dass die Verstöße „EXT.023“, „EXT.107“, „EXT.150“, „EXT.151“ und „EXT.135“ im Gegensatz zur Abbildung 4.20 hier komplett nicht vorhanden sind. Der Verstoß „EXT.101“, der vor der Anwendung BPMNspector-fixSeqFlow in 59% der BPMN-Prozessen mit Verstößen gefunden wurde, beträgt in der Abbildung 4.23 nur 1%.

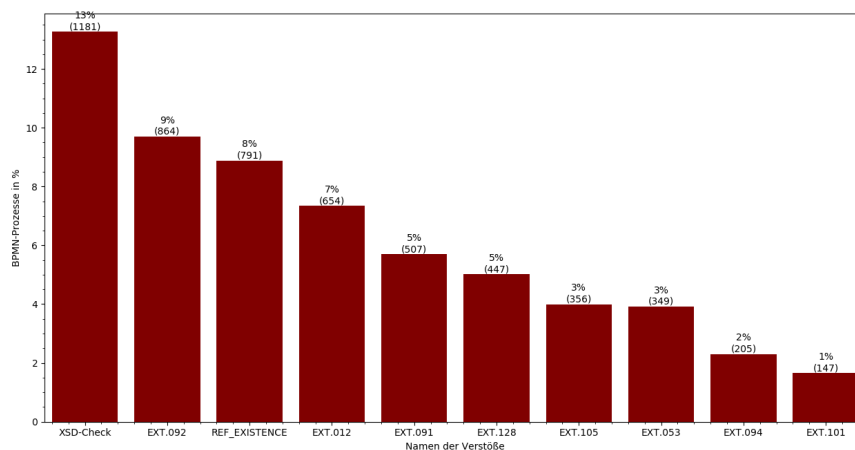


Abbildung 4.23.: BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen nach der Ausführung BPMNspector-fixSeqFlow (insgesamt 4.106 BPMN-Prozesse)

Die Tabelle 4.3 zeigt uns die Top-10 Verstöße gegen BPMN 2.0-Einschränkungen nach der Anwendung von BPMNspector-fixSeqFlow, aufsummiert über alle 4.106 BPMN-Prozesse mit Verstößen.

Wenn man die Abbildung 4.21 und Tabelle 4.3 vergleicht, dann ist es offensichtlich, dass die mit dem BPMNspector-fixSeqFlow reparierten Verstöße in der Tabelle 4.3 nicht mehr vorkommen. Interessant ist jedoch, dass die Anzahl der Verstöße „XSD-

	Absolute Häufigkeit	Relative Häufigkeit	Verstoß
1	44.230	67.63%	„XSD-Check“
2	6.788	10,38%	„EXT.092“
3	3.730	5,70%	„EXT.091“
4	1.524	2,33%	„REF_EXISTENCE“
5	1.015	1,55%	„EXT.063“
6	949	1,45%	„EXT.012“
7	790	1.21%	„EXT.128“
8	611	0,93%	„EXT.053“
9	456	0,70%	„EXT.105“
10	419	0,64%	„EXT.013“
11	4.889	7,48%	Andere

Tabelle 4.3.: Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 65.401)

Check“ von 43.516 (vor der Anwendung BPMNspector-fixSeqFlow) auf 44.230 (nach der Anwendung dieses Programms) gestiegen ist. Die Anzahl der Verstöße „REF_EXISTENCE“, „EXT.091“ und „EXT.092“ wurden fast nicht geändert.

5. Fazit

Im Rahmen dieser Arbeit wurde ein Programm für die Suche und Analyse von BPMN-Prozessen auf GitHub entwickelt. Mit diesem Programm wurden 10% von allen öffentlichen GitHub-Repositorys (6.163.217 GitHub-Repositorys) nach BPMN-Prozessen durchsucht und bei 1.251 Repositorys (0,02%) wurden BPMN-Prozesse gefunden. Dabei wurde eine Datei in einem Repository als BPMN-Prozess identifiziert, falls sie das Wort „bpmn“ im Namen enthält. Dies traf auf 21.306 Dateien zu. Eine weitere Analyse der Dateien auf ihren Inhalt ergab, dass es sich bei 16.907 Dateien von diesen um Dateien im XML-Format handelte, also überwiegend um XML-serialisierte BPMN-Prozesse.

Die weitere Analyse der ermittelten GitHub-Repositorys mit BPMN-Prozessen hat uns zu folgenden Erkenntnissen geführt:

- Die Mehrheit der Repositorys (51,5%) hat Java als dominante Programmiersprache. Dies deutet darauf hin, dass die Implementierung von Geschäftsprozessen sehr häufig mit Java erfolgt.
- Das Erscheinen der Version 2.0 von BPMN im Jahr 2011 bedeutete einen großen Schub für die Popularität von BPMN auf GitHub. So beträgt der jährliche Zuwachs an Repositorys mit BPMN-Prozessen in den Jahren 2011 bis 2017 durchschnittlich 71,7%.
- Die durchschnittliche Dauer der Aktivität in allen Repositorys mit BPMN-Dateien beträgt 7 Monate und 17 Stunden und durchschnittliche Anzahl der Commits ist 171.
- Es deutet viel darauf hin, dass China, die Vereinigten Staaten von Amerika und Deutschland die Standorte mit den meisten Autoren von GitHub-Repositorys mit BPMN-Prozessen sind.

Die wichtigsten Erkenntnisse aus der Analyse von BPMN-Prozessen im XML-Format sind die Folgenden:

- Die Mehrheit der gefundenen BPMN-Prozesse (91,3%) auf GitHub hat nur einen Autor.
- 38,9% der gefundenen BPMN-Prozesse sind nicht älter als ein Jahr.
- Die Mehrheit der gefundenen BPMN-Prozesse (73%) wurde nach ihrer Erstellung nicht mehr geändert.
- Der Anteil der Duplikate unter den gefundenen BPMN-Prozesse ist relativ groß und beträgt 47,3%.

Die Anwendung des Analysewerkzeugs BPMNspector auf BPMN-Prozesse im

XML-Format ergab, dass nur ein kleiner Anteil der Prozesse (1.471 der gefundenen Prozesse ohne Duplikate) keinen Regelverstoß gegenüber der BPMN-Spezifikation aufweist. In der überwiegenden Zahl der Prozesse (7.376 der gefundenen Prozesse ohne Duplikate) hat BPMNspector hingegen mindestens ein Verstoß gegen eine Syntax- oder Semantikregel identifizieren können. Mit der Anwendung des Reparaturwerkzeug BPMNspector-fixSeqFlow konnten 3.270 von 7.376 BPMN-Prozessen mit Verstößen vollständig repariert werden.

Anhand dieser Zahlen wird ersichtlich, dass es offenbar einen großen Bedarf an Analysewerkzeugen für BPMN, wie beispielsweise BPMNspector, gibt, um Modellierungs- und Programmierfehler zu identifizieren und zu vermeiden.

6. Ausblick

Bei der Durchführung unserer Arbeit wurden einige Aspekte nicht vollständig untersucht und können noch weiter erforscht werden:

- So kann GitHub vollständig nach BPMN-Prozessen durchsucht werden und nicht nur 10% der Repositories wie in dieser Arbeit. Dafür kann das von uns entwickelte Mining-Programm benutzt werden, das die parallele Ausführung an einem oder an mehreren Rechnern ermöglicht und auf solche Weise das Durchsuchen von GitHub erheblich beschleunigen kann.
- Die Identifikation der BPMN-Prozesse im XML-Format kann noch verbessert werden. Es kann nicht nur nach dem Wort „bpmn“ im Dateinamen gesucht werden (wie in unserem Mining-Programm), sondern auch der Inhalt der Datei durchsucht werden. Es können zum Beispiel alle XML-Dateien nach der Sequenz „<http://www.omg.org/spec/BPMN/20100524/MODEL>“ durchsucht werden, die das Wort „bpmn“ im Dateinamen nicht haben. Zwar wird bei diesem Szenario die Geschwindigkeit des Mining-Prozesses deutlich langsamer, aber die Präzision der Suche wird dadurch erheblich höher.
- Es kann die Identifikation und Analyse der BPMN-Prozesse, die als Bilder gespeichert sind, implementiert werden. Dafür kann z.B. die Erkennung der BPMN-Elemente innerhalb einer Bilddatei programmiert werden.
- Die Berechnung der Größe von BPMN-Prozessen im XML-Format kann dadurch präzisiert werden, dass statt XML-Elementen eines Prozesses die BPMN-Elemente gezählt werden. Dafür muss aber ein Programm geschrieben werden, das alle in der BPMN-Spezifikation definierten BPMN-Elemente in einem BPMN-Prozess erkennen kann.

Literaturverzeichnis

- [1] Bpmn 2.0-symbolreferenz. <https://camunda.com/bpmn/reference/>. Accessed: 2019-06-24.
- [2] Bpmn-datei: Historicprocessinstancecollectionresource-test.testqueryprocessinstances.bpmn20.xml. <https://raw.githubusercontent.com/raoifeng/activiti-explorer/master/src/test/resources/org/activiti/rest/service/api/history/HistoricProcessInstanceCollectionResourceTest.testQueryProcessInstances.bpmn20.xml>. Accessed: 2019-07-07.
- [3] Bpmn-datei: Historictaskinstanceupdatetest.testhistorictaskinstanceupdate.bpmn20.xml. <https://raw.githubusercontent.com/249134995/ace/master/ace/activiti-root/modules/activiti-engine/src/test/resources/org/activiti/engine/test/history/HistoricTaskInstanceUpdateTest.testHistoricTaskInstanceUpdate.bpmn20.xml>. Accessed: 2019-07-07.
- [4] Bpmn-datei: leave.bpmn. <https://raw.githubusercontent.com/11653786/jeesite/master/src/main/resources/act/designs/oa/leave/leave.bpmn>. Accessed: 2019-07-07.
- [5] Bpmn-datei: test_audit.bpmn. https://raw.githubusercontent.com/11653786/jeesite/master/src/main/resources/act/designs/oa/test_audit/test_audit.bpmn. Accessed: 2019-07-07.
- [6] bpmn-moddle-auto-layout. <https://github.com/bpmn-io/bpmn-moddle-auto-layout>. Accessed: 2019-07-07.
- [7] Bpmn-prozess: Intermittentqradstatus.bpmn. <https://raw.githubusercontent.com/themarkler/bpmrepo/master/fsr-diagnostic-engine-kjar/src/main/resources/IntermittentQRADStatus.bpmn>. Accessed: 2019-07-07.
- [8] Bpmn-prozess: Modemauthlensemblepppoe.bpmn. <https://raw.githubusercontent.com/themarkler/bpmrepo/master/fsr-diagnostic-engine-kjar/src/main/resources/ModemAuthLEnsemblePPPoE.bpmn>. Accessed: 2019-07-07.
- [9] Bpmn-prozess: Oao.bpmn2. <https://raw.githubusercontent.com/jstoeckelcrossvale/OAO/master/OAO/src/main/resources/com/crossvale/oao/OAO.bpmn2>. Accessed: 2019-07-07.
- [10] Bpmn-prozess: Slowspeedrange.bpmn. <https://raw.githubusercontent.com/themarkler/bpmrepo/master/fsr-diagnostic-engine-kjar/src/main/resources/SlowSpeedRange.bpmn>. Accessed: 2019-07-07.

- [11] Bpmn-prozess: usertasks.bpmn. <https://raw.githubusercontent.com/kiegroup/kie-wb-common/master/kie-wb-common-stunner/kie-wb-common-stunner-sets/kie-wb-common-stunner-bpmn/kie-wb-common-stunner-bpmn-backend/src/test/resources/org/kie/workbench/common/stunner/bpmn/backend/service/diagram/userTasks.bpmn>. Accessed: 2019-07-07.
- [12] Bpmn workflow engine. <https://camunda.com/de/products/bpmn-engine/>. Accessed: 2019-06-21.
- [13] Bpmn_crawler. https://github.com/ViktorStefanko/BPMN_Crawler/. Accessed: 2019-06-28.
- [14] Bpmnspector-fixseqflow. <https://github.com/matthiasgeiger/BPMNspector-fixSeqFlow>. Accessed: 2019-06-11.
- [15] Bpmnspector. static analysis of bpmn 2.0 process models. <http://bpmnspector.org/>. Accessed: 2019-05-12.
- [16] Business process model and notation. <https://www.omg.org/spec/BPMN/2.0.2>. Accessed: 2019-04-15.
- [17] Camunda modeler. <https://camunda.com/de/products/modeler/>. Accessed: 2019-06-21.
- [18] Code maat. <https://github.com/adamtornhill/code-maat>. Accessed: 2019-07-02.
- [19] Constraintschecker. https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/bpmn_spector/run_bpmn_spector.py. Accessed: 2019-07-07.
- [20] Constraintsfixer. https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/BPMNspector-fixSeqFlow/ConstraintsFixer.py. Accessed: 2019-07-07.
- [21] Duplicate files finder. <http://doubles.sourceforge.net/>. Accessed: 2019-05-12.
- [22] FERNANDEZ, M. A., AND ROBLES, G. Extracting software development information from floss projects in github. Paper at SATToSE 2017, http://sattose.wdfiles.com/local--files/2017:schedule/SATToSE_2017_paper_6.pdf, 2017. Accessed: 2019-04-20.
- [23] Filestatistics. https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/statistics/file_statistics.py. Accessed: 2019-07-07.
- [24] GEIGER, M. BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, no. 92, Otto-Friedrich Universität Bamberg, May 2013.
- [25] Ghtorrent on the google cloud. <http://ghtorrent.org/gcloud.html>. Accessed: 2019-04-19.

- [26] Github. <https://github.com/>. Accessed: 2019-04-15.
- [27] Github facts. <https://github.com/about/facts>. Accessed: 2019-04-19.
- [28] Githut 2.0. https://madnight.github.io/githut/#/pull_requests/2019/1. Accessed: 2019-07-08.
- [29] Gitlab. <https://about.gitlab.com/>. Accessed: 2019-04-15.
- [30] Google bigquery. <https://cloud.google.com/bigquery/>. Accessed: 2019-04-19.
- [31] GOUSIOS, G., VASILESCU, B., SEREBRENIK, A., AND ZAIDMAN, A. Leantorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (New York, NY, USA, 2014), MSR 2014, ACM, pp. 384–387.
- [32] HEBIG, R., QUANG, T. H., CHAUDRON, M. R. V., ROBLES, G., AND FERNANDEZ, M. A. The quest for open source projects that use uml: Mining github. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (New York, NY, USA, 2016), MODELS '16, ACM, pp. 173–183.
- [33] HERZIG, K. S., AND ZELLER, A. Mining your own evidence. In *Making Software - What Really Works, and Why We Believe It*. O'Reilly Media, Inc., Sebastopol, 2011, pp. 517–529.
- [34] KALE, V. *Enterprise Process Management Systems - Engineering Process-Centric Enterprise Systems using BPMN 2.0*. CRC Press, Boca Raton, Fla, 2018.
- [35] KOCIAN, C. *Geschäftsprozessmodellierung mit BPMN 2.0 Business Process Model and Notation im Methodenvergleich*. HNU Working Paper Nr. 16. Neu-Ulm University of Applied Sciences, Wileystraße 1, D-89231 Neu-Ulm, 2011.
- [36] Object management group. <https://www.omg.org/>. Accessed: 2019-04-15.
- [37] OMG. *Business Process Model and Notation (BPMN)*. Needham USA(2013), 2013.
- [38] Places with the most contributors. <https://octoverse.github.com/people>. Accessed: 2019-05-22.
- [39] Prozessmodellierung mit epk. <http://www.leed.ch/history/eeepk/>. Accessed: 2019-04-15.
- [40] Repositorystatistics. https://github.com/ViktorStefanko/BPMN_Crawler/blob/master/scripts/statistics/repository_statistics.py. Accessed: 2019-07-07.
- [41] Rest api v3. <https://developer.github.com/v3/>. Accessed: 2019-04-22.
- [42] ROBLES, G. 2016-uml-miner. <https://github.com/LibreSoftTeam/2016-uml-miner>. Accessed: 2019-04-23.

- [43] ROBLES, G., HO-QUANG, T., HEBIG, R., CHAUDRON, M., AND FERNANDEZ, M. An extensive dataset of uml models in github. pp. 519–522.
- [44] ROBLES, G., HO-QUANG, T., HEBIG, R., CHAUDRON, M. R. V., AND FERNANDEZ, M. A. An extensive dataset of uml models in github. In *Proceedings of the 14th International Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2017), MSR '17, IEEE Press, pp. 519–522.
- [45] SANTACROCE, F., OLSSON, A., VOSS, R., AND NAREBSKI, J. *Git: Mastering Version Control*. Packt Publishing Ltd, Birmingham, 2016.
- [46] The ghtorrent project. <http://ghtorrent.org/>. Accessed: 2019-04-19.
- [47] The state of the octoverse. <https://octoverse.github.com/>. Accessed: 2019-04-19.
- [48] TORNHILL, A. *Your Code as a Crime Scene - Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs*. Pragmatic Bookshelf, Raleigh, North Carolina and Dallas, Texas, 2015.
- [49] Übersicht zur prozessmodellierungssprache bpmn 2.0. <https://www.processhub.com/uebersicht-zur-prozessmodellierungssprache-bpmn-2-0/>. Accessed: 2019-06-07.
- [50] Unified modeling language. <https://www.uml.org/>. Accessed: 2019-04-15.

Abbildungsverzeichnis

2.1. Camunda Modeler (grafischer Editor)	9
2.2. Camunda Modeler (Texteditor)	9
2.3. Buchausleihe	14
2.4. Algorithmus für das MSR	16
3.1. MySQL-Datenbankschema der Tabellen „users“ und „projects“ . . .	18
3.2. Beispiel von einem zurückgelieferten JSON-Objekt der Suchschnitt- stelle der GitHub API	20
3.3. Datenbankschema	22
3.4. Prozessdiagramm des Mining-Programms	24
3.5. Klassendiagramm des Mining-Programms	25
3.6. Ein einfacher BPMN-Prozess	29
3.7. Ein einfacher BPMN-Prozess mit Verletzung einer <sequenceFlow> Einschränkung. Übernommen aus [14].	29
3.8. Ein einfacher BPMN-Prozess ohne Verletzung einer <sequence- Flow> Einschränkung. Übernommen aus [14].	29
4.1. Dateiformate von potentiellen BPMN-Prozessen (gruppiert)	32
4.2. Dominante Programmiersprachen der Repositorys mit BPMN-Dateien	33
4.3. Dominante Programmiersprachen von allen öffentlichen GitHub- Repositorys, erstellt im 1. Quartal 2019 [28]	34
4.4. Jahre, in denen Repositorys mit BPMN-Dateien erstellt wurden . . .	34
4.5. Jahre, in denen Repositorys mit BPMN-Dateien letztes Mal geändert wurden	35
4.6. Jahre, in denen Repositorys mit BPMN-Dateien erstellt und letztes Mal geändert wurden	36
4.7. Dauer der Aktivitäten von GitHub-Repositorys mit BPMN-Dateien .	36
4.8. Anzahl der Commits von GitHub-Repositorys mit BPMN-Dateien . .	37
4.9. Standorte der Repository-Entwickler	38
4.10. Standorte mit den meisten GitHub-Benutzer [38]	38
4.11. Anteil BPMN-Dateien, die dupliziert wurden	39
4.12. Statistik über Duplikate von BPMN-Dateien	39
4.13. Beispiel eines BPMN-Prozesses im XML-Format	40
4.14. Anteil der BPMN-Prozesse im XML-Format mit und ohne das XML- Element BPMNDiagramm	41
4.15. Größe der BPMN-Prozesse im XML-Format	42
4.16. Anzahl der Autoren von BPMN-Dateien	42
4.17. Alter in Jahren von BPMN-Dateien	43

4.18. Anzahl der Änderungen von BPMN-Dateien	43
4.19. Anteil valider und nicht-valider BPMN-Prozesse	44
4.20. BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen (insgesamt 7.376 BPMN-Prozesse)	45
4.21. Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 150.168 Verstöße)	46
4.22. Anteil valider und nicht-valider BPMN-Prozesse nach der Anwen- dung BPMNspector-fixSeqFlow	48
4.23. BPMN-Prozesse mit Verstößen gegen BPMN 2.0-Einschränkungen nach der Ausführung BPMNspector-fixSeqFlow (insgesamt 4.106 BPMN-Prozesse)	48
A.1. BPMN-Datei: <code>HistoricProcessInstanceCollection</code> <code>ResourceTest.testQueryProcessInstances.bpmn20.xml</code> [2]. Wurde 89-Mal dupliziert.	60
A.2. BPMN-Datei: <code>test_audit.bpmn</code> [5]. Wurde 67-Mal dupliziert.	60
A.3. BPMN-Datei: <code>leave.bpmn</code> [4]. Wurde 60-Mal dupliziert.	61
A.4. BPMN-Datei: <code>HistoricTaskInstanceUpdateTest.</code> <code>testHistoricTaskInstanceUpdate.bpmn20.xml</code> [3]. Wurde 48-Mal dupliziert.	61
B.1. BPMN-Prozess: <code>IntermittentQRADStatus.bpmn</code> [7]. Besteht aus 2.693 XML-Elementen.	63
B.2. BPMN-Prozess: <code>OA0.bpmn</code> [9]. Besteht aus 2.630 XML-Elementen.	64
B.3. BPMN-Prozess: <code>userTasks.bpmn</code> [11]. Besteht aus 2.284 XML- Elementen.	65
B.4. BPMN-Prozess: <code>ModemAuthLEnsemblePPPoE.bpmn</code> [8]. Besteht aus 2.243 XML-Elementen.	66
B.5. BPMN-Prozess: <code>SlowSpeedRange.bpmn</code> [10]. Besteht aus 2.175 XML-Elementen.	67

Tabellenverzeichnis

2.1. Grundlegende Modellierungselemente von BPMN-Prozessen [16], [35], [1]	10
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen	11
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen	12
2.1. Grundlegende Modellierungselemente von BPMN-Prozessen	13
4.1. Resultate des Mining-Programms	32
4.2. Reparierte Verstöße gegen BPMN 2.0-Einschränkungen	47
4.3. Top-10 Verstöße gegen BPMN 2.0-Einschränkungen (insgesamt 65.401)	49
C.1. Verstöße gegen BPMN 2.0-Einschränkungen	68
C.1. Verstöße gegen BPMN 2.0-Einschränkungen	69
C.1. Verstöße gegen BPMN 2.0-Einschränkungen	70

A. Anhang

A.1. Die 4 am meisten duplizierten BPMN-Dateien im XML-Format

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions
3   xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
4   xmlns:activiti="http://activiti.org/bpmn"
5   targetNamespace="OneTaskCategory">
6
7   <process id="oneTaskProcess" name="The One Task Process">
8     <documentation>One task process description</documentation>
9
10    <startEvent id="theStart" />
11    <sequenceFlow id="flow1" sourceRef="theStart" targetRef="processTask" />
12    <userTask id="processTask" name="Process task" activiti:assignee="kermit">
13      <documentation>Process task description</documentation>
14    </userTask>
15    <sequenceFlow id="flow2" sourceRef="processTask" targetRef="processTask2" />
16    <userTask id="processTask2" name="Process task2" activiti:assignee="fozzie">
17      <documentation>Process task description2</documentation>
18    </userTask>
19    <sequenceFlow id="flow3" sourceRef="processTask2" targetRef="theEnd" />
20    <endEvent id="theEnd" />
21
22  </process>
23
24 </definitions>
```

Abbildung A.1.: BPMN-Datei: HistoricProcessInstanceCollection
ResourceTest.testQueryProcessInstances.bpmn20.xml
[2]. Wurde 89-Mal dupliziert.

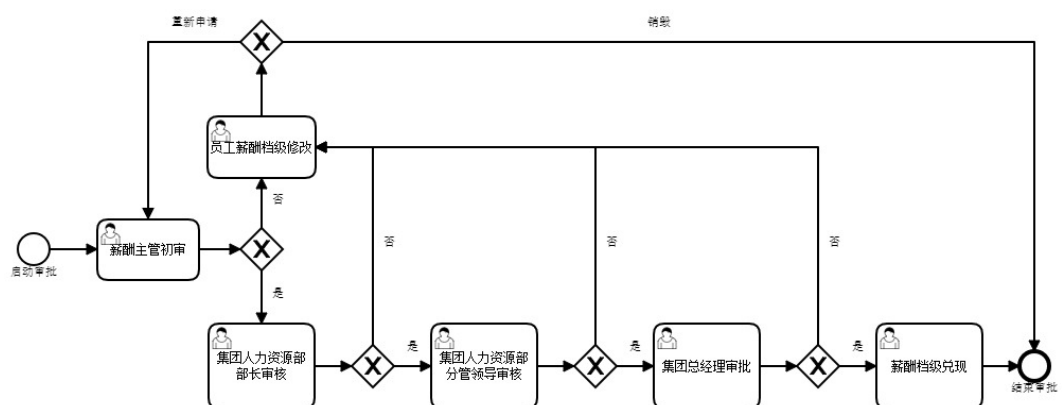


Abbildung A.2.: BPMN-Datei: test_audit.bpmn [5]. Wurde 67-Mal dupliziert.

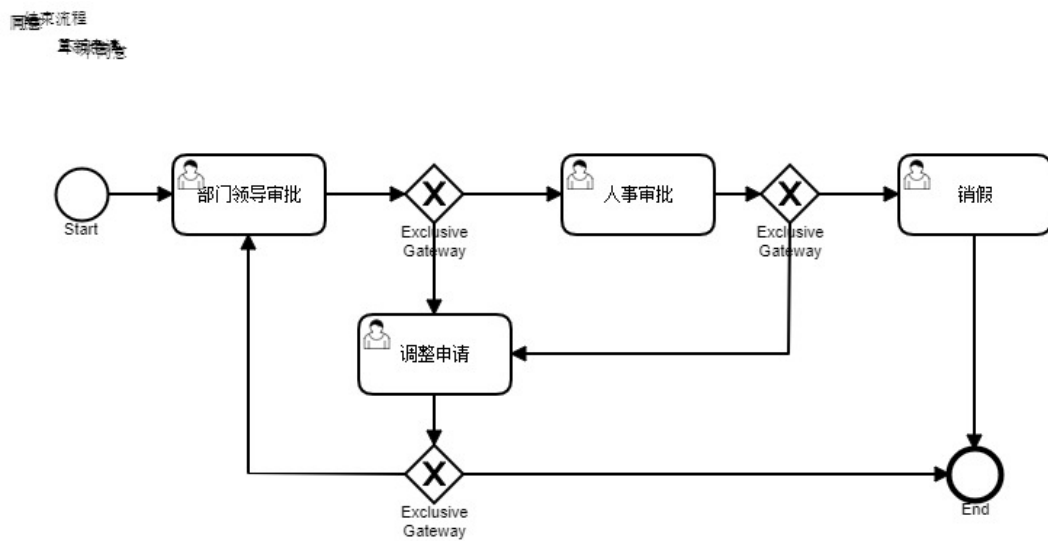


Abbildung A.3.: BPMN-Datei: leave.bpmn [4]. Wurde 60-Mal dupliziert.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <definitions id="taskAssigneeExample"
3   xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
4   xmlns:camunda="http://camunda.org/schema/1.0/bpmn"
5   targetNamespace="Examples">
6
7   <process id="HistoricTaskInstanceTest">
8
9     <startEvent id="start" />
10
11     <sequenceFlow id="flow1" sourceRef="start" targetRef="task" />
12
13     <userTask id="task" name="Clean up">
14       <documentation>
15         Schedule an engineering meeting for next week with the new hire.
16       </documentation>
17       <humanPerformer>
18         <resourceAssignmentExpression>
19           <formalExpression>kermit</formalExpression>
20         </resourceAssignmentExpression>
21       </humanPerformer>
22     </userTask>
23
24     <sequenceFlow id="flow2" sourceRef="task" targetRef="end" />
25
26     <endEvent id="end" />
27
28   </process>
29
30 </definitions>

```

Abbildung A.4.: BPMN-Datei: HistoricTaskInstanceUpdateTest.
testHistoricTaskInstanceUpdate.bpmn20.xml [3].
Wurde 48-Mal dupliziert.

B. Anhang

B.1. Die 5 größten BPMN-Prozesse im XML-Format

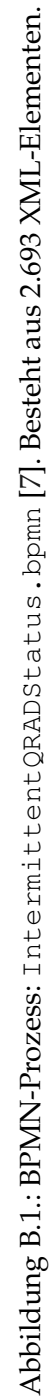


Abbildung B.1.: BPMN-Prozess: IntermittentQRADStatus.bpmn [7]. Besteht aus 2.693 XML-Elementen.

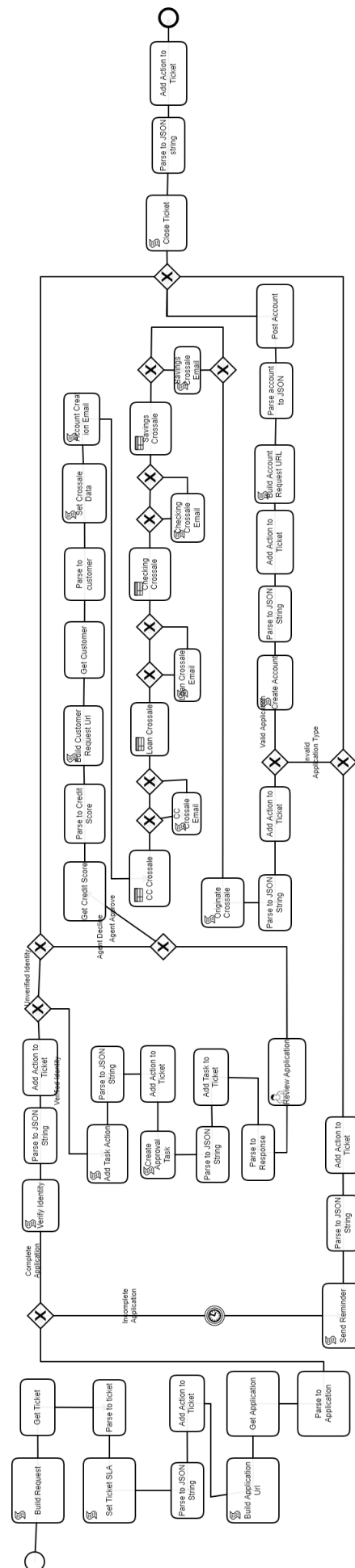


Abbildung B.2.: BPMN-Prozess: OA0.bpmn [9]. Besteht aus 2.630 XML-Elementen.

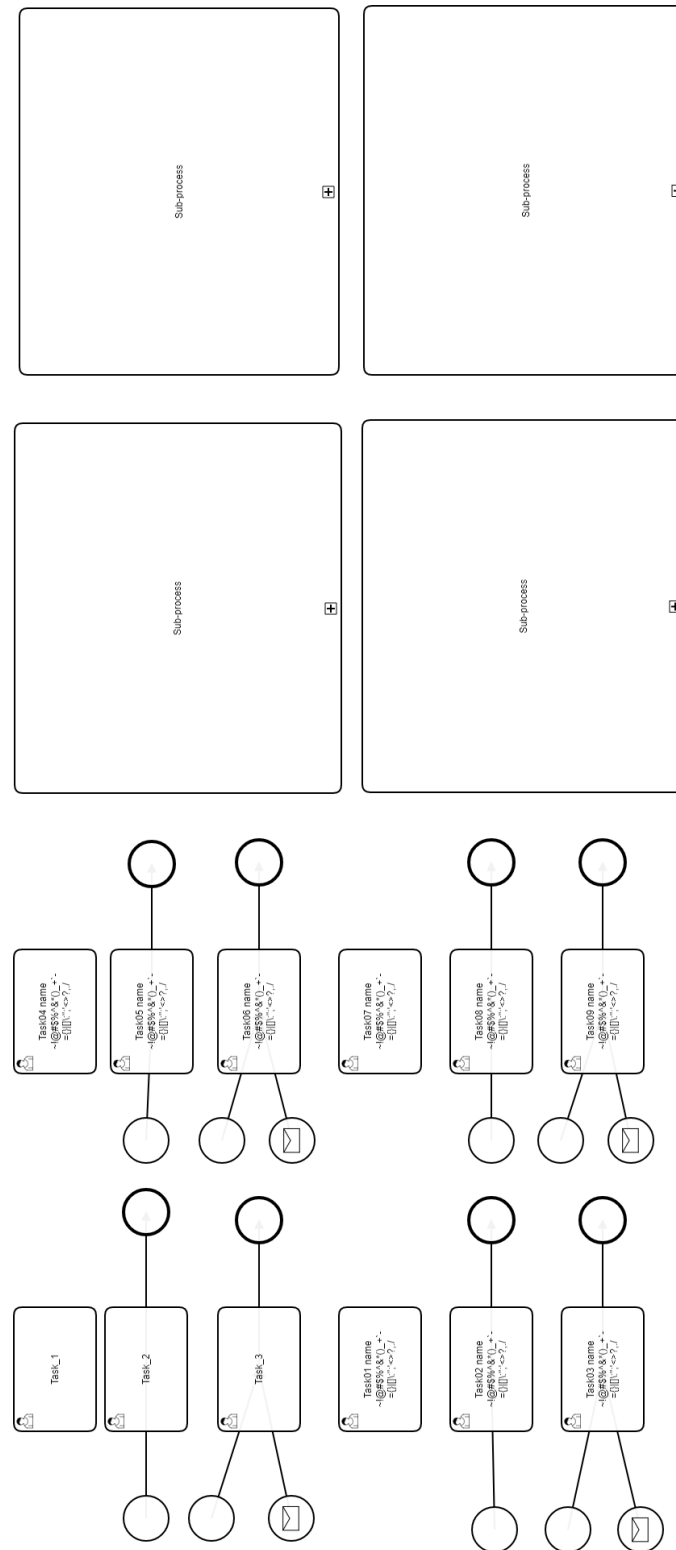


Abbildung B.3.: BPMN-Prozess: userTasks.bpmn [11]. Besteht aus 2.284 XML-Elementen.

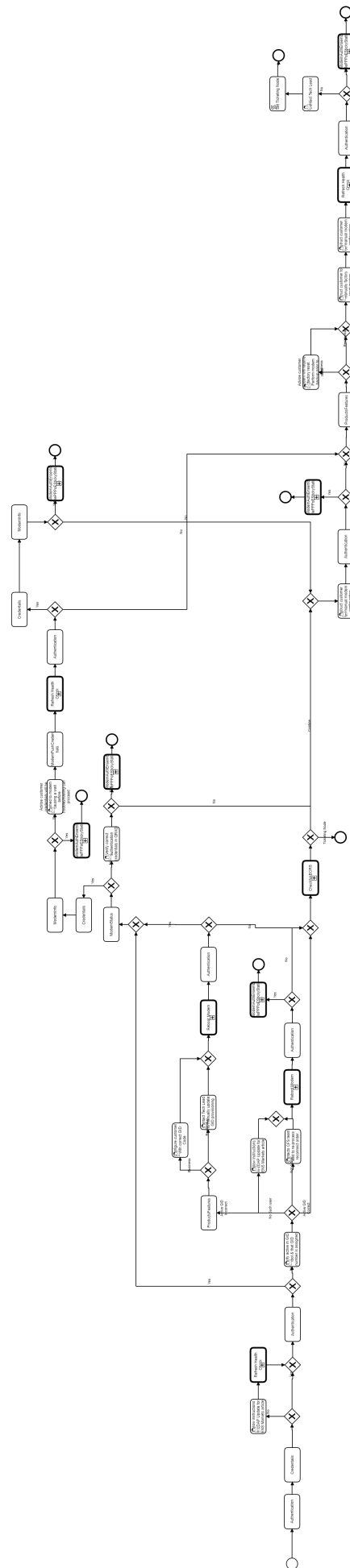


Abbildung B.4.: BPMN-Prozess: ModemAuthLEnsemble.ePPPoE.bpmn [8]. Besteht aus 2.243 XML-Elementen.

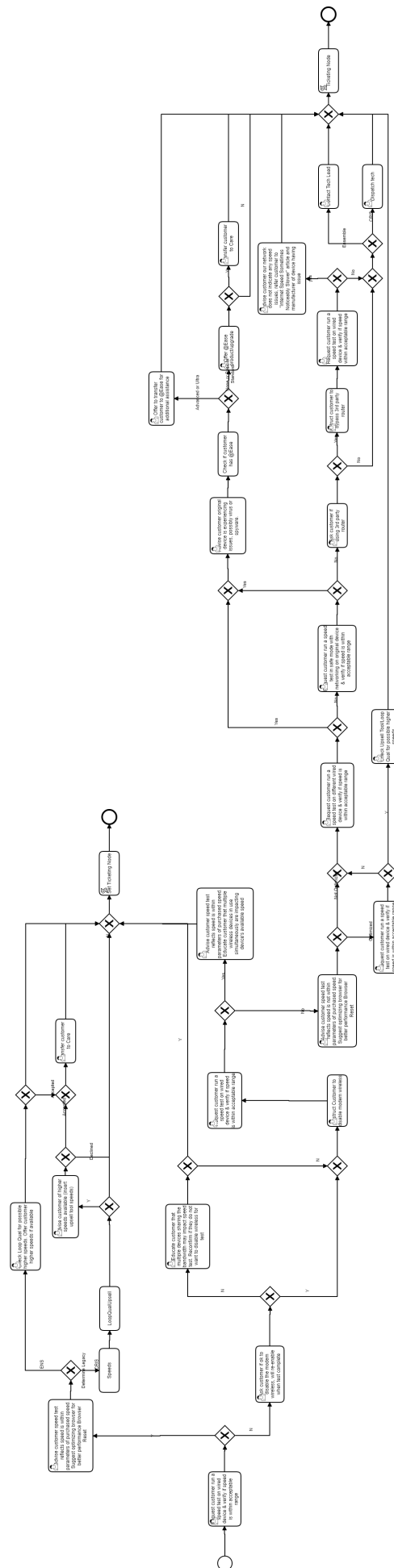


Abbildung B.5.: BPMN-Prozess: SlowSpeedRange.bpmn [10]. Besteht aus 2.175 XML-Elementen.

C. Anhang

C.1. Typen von Verstößen gegen die Syntax- und Semantikregeln von BPMN 2.0

Tabelle C.1.: Verstöße gegen BPMN 2.0-Einschränkungen

	Name des Verstoßes	Betroffenes Element	Bedeutung
1	EXT.012	<Expression>	Wenn Ausdrücke in natürlicher Sprache verwendet werden, kann der Prozess nicht ausgeführt werden [24].
2	EXT.013	<Formal Expression>	Formeller Rumpf muss vorhanden sein [24].
3	EXT.023	<Sequence Flow>	Das Quell- und Zielement des Sequenzflusses müssen unter Verwendung der eingehenden oder ausgehenden Attribute auf die SequenceFlow-Definition verweisen [24].
4	EXT.053	<Script Task>	Wenn ein Skript vorhanden ist, muss der Skripttyp definiert werden [24].
5	EXT.063	<Call Activity>	Elemente, die in der InputOutputSpecification der Aufrufaktivität enthalten sind, müssen genau mit den Elementen übereinstimmen, die im referenzierten CallableElement enthalten sind. Dies umfasst DataInputs, DataOutputs, InputSets und OutputSets [24].
6	EXT.067	<Start Event>	Wenn keine Transformation definiert oder referenziert ist, muss nur eine Quelle definiert werden [24].
7	EXT.084	<Data Input>	Ein Dateneingang muss von mindestens einem Eingabesatz referenziert werden [24].
8	EXT.091	<Data Association>	sourceRef und targetRef müssen dieselbe ItemDefinition haben, oder es muss eine Transformation vorhanden sein [24].

Tabelle C.1.: Verstöße gegen BPMN 2.0-Einschränkungen

	Name des Versto- ßes	Betroffenes Element	Bedeutung
9	EXT.092	<Data As- sociation>	Wenn keine Transformation definiert oder referen- ziert ist, muss nur eine Quelle definiert werden [24].
10	EXT.094	<Event>	Für jede Ereignisdefinition muss eine Element- definition mit der zugehörigen Dateneingabe / - ausgabe vorhanden sein [24].
11	EXT.101	<Start Event>	Der Startereignis muss die Quelle für den Se- quenzfluss sein [24].
12	EXT.105	<End Event>	Ein Endereignis muss vorhanden sein, wenn ein Startereignis auf derselben Prozessebene verwen- det wird [24].
13	EXT.107	<End Event>	Das Endereignis musss mindestens einen einge- henden Sequenzfluss haben [24].
14	EXT.113	<Boundary Event>	Ein Grenzerignis muss mindestens eine Quelle für einen Sequenzfluss sein [24].
15	EXT.115	<Intermedi- ate Event>	Zwischenereignis enthält keinen Link. Zwische- nereignisse müssen ein Ziel von mindestens einem Sequenzfluss sein [24].
16	EXT.116	<Intermedi- ate Event>	Zwischenereignis enthält keinen Link. Zwische- nereignisse müssen eine Quelle für mindestens einen Sequenzfluss sein [24].
17	EXT.128	<Message Event De- finition>	Ein operationRef muss vorhanden sein, wenn der Prozess ausführbar sein soll [24].
18	EXT.135	<Gateway>	Ein Gateway muss entweder mehrere eingehende Sequenzflüsse oder mehrere ausgehende Sequenz- flüsse haben [24].
19	EXT.136	<Event Based Gateway>	Ein Ereignis-Gateway muss zwei oder mehr aus- gehende Sequenzflüsse haben [24].
20	EXT.150	<Start Event>	Wenn ein Startereignis zum Initiieren eines Pro- zesses verwendet wird, müssen alle Ablaufkno- ten (außer Startereignissen, Grenzerignissen und abfangenden Verknüpfungserignissen, Kompen- sationsaktivitäten und Ereignis-Unterprozessen) einen eingehenden Ablauf haben [24].

Tabelle C.1.: Verstöße gegen BPMN 2.0-Einschränkungen

	Name des Versto- ßes	Betroffenes Element	Bedeutung
21	EXT.151	<End Event>	Wenn Endereignisse verwendet werden, müs- sen alle anderen Ablaufknoten (außer Ender- eignissen, auslösenden Verknüpfungseignissen, Ausgleichsaktivitäten und Ereignisunterprozes- sen) einen ausgehenden Ablauf haben [24].
22	XSD- Check	-	Fehler in Dateistruktur.
23	REF_EX ISTEN CE	-	Wenn ein BPMN-Element auf ein anderes Element verweist, muss die Referenz auflösbar sein. D.h. das referenzierte Element muss vorhanden sein. Darüber hinaus dürfen in den meisten Fällen nur bestimmte Elemente referenziert werden [24].